Profesor:

Jesse Padilla Agudelo

Integrantes:

Brenda Catalina Barahona Pinilla (201812721)
Juan Diego González Gomez (201911031)
Kevin Steven Gamez Abril (201912514)
Sergio Julian Zona Moreno (201914936)

Tabla de contenido

1 Introducción

En este documento se documenta la solución a la Entrega 1 del proyecto del curso *Desarrollo de soluciones cloud*. El propósito central es desplegar una aplicación Web completa (Frontend + Backend + Base de datos relacional + Servicios asincrónicos) utilizando contenedores de Docker, y *Docker Compose* para desplegarla en una instancia virtual EC2 de AWS.

2 Arquitectura de Software

2.1 Diagrama de despliegue y descripción

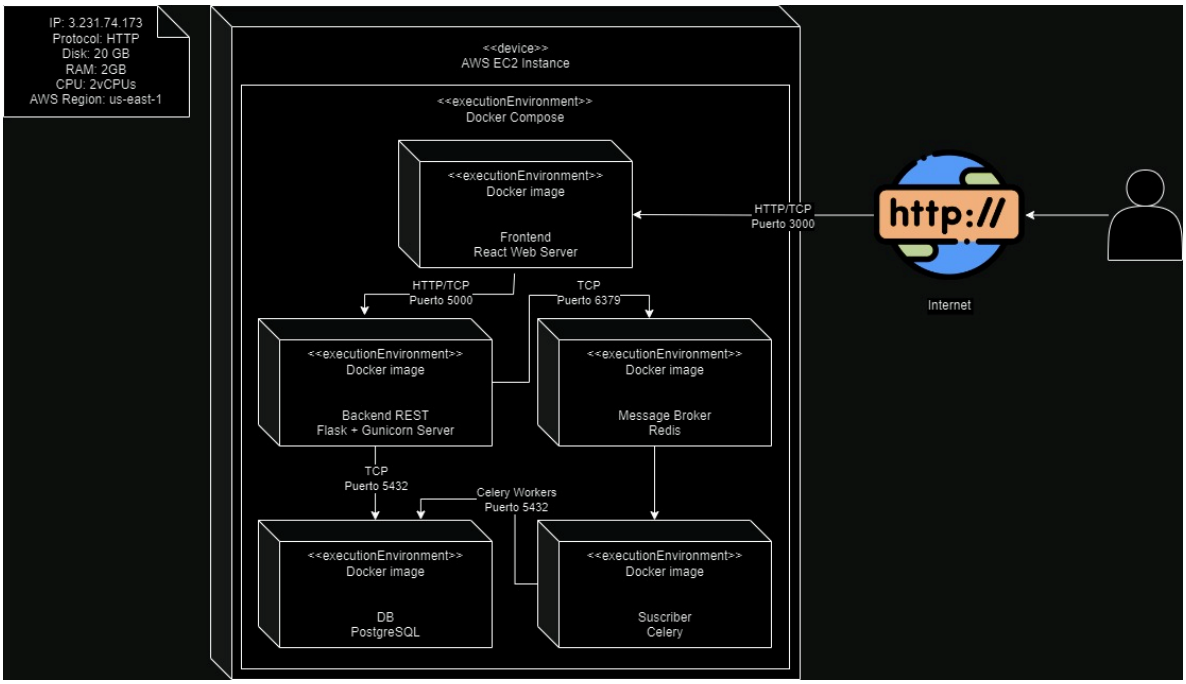

Figura 1. Diagrama de despliegue de la aplicación.

La arquitectura de Software es bastante sencilla. En primera instancia, el usuario se conecta a internet y por medio de la IP pública de la instancia EC2 de AWS donde puede acceder al servicio de Frontend de la aplicación. Este servicio, a su vez se alimenta del servicio de Backend que se encuentra involucrado con otros tres servicios: la base de datos de PostgreSQL, Redis y Celery. Redis y Celery son quienes permiten el funcionamiento asincrónico del backend, siendo Redis el Message Broker al cuál se suscribe Celery, el cual posteriormente inicia los Workers necesarios para procesar los archivos y actualizar la base de datos de PostgreSQL. De manera paralela, el Backend tiene una conexión directa con la base de datos para poder extraer la información de manera rápida y enviarla al cliente en el Frontend. Los archivos originales y procesados en PDFs se almacenan en una carpeta en el Backend, por lo que dichos archivos se encuentran en el volumen donde se clona el repositorio de la instancia EC2.

2.2 Instrucciones para la ejecución en local

- Clonar el repositorio https://github.com/JuanDiegoGonzalez/Proyecto_Cloud_Grupo2
- Desde una terminal, en la carpeta raíz del repositorio, ejecutar los comandos:
  - docker-compose build
  - docker-compose up –d

- Ingresar a la ruta: http://localhost:3000/ (NOTA: si la página no carga, o si aparece el error “This page isn’t working. localhost didn’t send any data. ERR_EMPTY_RESPONSE”, esperar unos segundos mientras se termina de ejecutar el Frontend).

2.3 Release

https://github.com/JuanDiegoGonzalez/Proyecto_Cloud_Grupo2/releases/tag/1.0

2.4 Vídeo sustentación

2.5 Documentación del API

https://documenter.getpostman.com/view/12924938/2sA2xb7voa

3 Consideraciones para escalar

Si bien en nuestra implementación no se desarrolló un balanceador de carga, es pertinente que, cuando se tiene una gran cantidad de usuarios, los servicios se puedan modular y distribuir de acuerdo con las capacidades de la máquina. Para ello, un balanceador de carga que redirija a los usuarios es una herramienta necesaria. Dependiendo de la carga prevista y de las capacidades de nuestra infraestructura, podríamos considerar implementar un balanceador de carga basado en software. Asimismo, explorar la posibilidad de migrar hacia una arquitectura de microservicios en el futuro podría ser factible, ya que estos pueden permitir una mayor escalabilidad, flexibilidad en el desarrollo y mantenimiento de la aplicación.

Otra alternativa importante, es tener una instancia que se pueda aprovisionar dinámicamente con escalamiento vertical, por lo que las características de esta, después de pasar cierto umbral, aumenta sus especificaciones para recibir al flujo masivo de usuarios. Además del escalamiento vertical, considerar la implementación de técnicas de escalamiento automático horizontal podría ser beneficioso. Esto implicaría agregar o eliminar instancias de nuestros servicios de manera automática en función de la carga de trabajo.

Dependiendo de la cantidad de datos y la frecuencia de acceso, podríamos necesitar utilizar bases de datos distribuidas o sistemas de almacenamiento en la nube que puedan escalar para manejar la carga. El uso de caché para almacenar datos temporales también puede ser útil para reducir la carga en la base de datos y mejorar el rendimiento de la aplicación.

Finalmente, sería prudente realizar pruebas de carga periódicas para identificar cuellos de botella y evaluar el rendimiento de la aplicación bajo diferentes niveles de carga. Implementar herramientas de monitoreo permitirían supervisar constantemente el rendimiento de la aplicación y la infraestructura.

4 Limitaciones

Dentro de las limitaciones podemos encontrar que:

1. Solo tenemos una instancia EC2 que recibe todas las peticiones. Si bien hay una cola de prioridad, es pertinente considerar implementar un balanceador de carga y más instancias EC2 que corran los servicios de Backend.
2. Para esta entrega en específico no se desarrollaron pruebas de carga en la aplicación. Tampoco existe un sistema de monitoreo del rendimiento de la aplicación.
3. No se implementaron mecanismos de escalamiento (ni vertical, ni horizontal).
4. La infraestructura de la aplicación está actualmente limitada a una única región de AWS, lo que podría resultar en vulnerabilidades ante posibles fallos o problemas de disponibilidad en esa región específica.
5. Dado que solo tenemos una instancia EC2 y un único Docker Compose, tenemos un único punto de falla. Adicionalmente, no se pueden gestionar los componentes de manera independiente (porque todos están dentro del Docker Compose).