Informe Proyecto Final

Jhojan Stiven Castaño Jejen -2259476

David Santiago Velazco Triana-2259479

William Alexander Franco Otero-2259715

Universidad del Valle

Ingeniería de Sistemas

Infraestructuras paralelas y Distribuidas

Sexto Semestre

Tuluá, Valle del Cauca

2024

1 Introducción

1.1 Descripción Del proyecto

El proyecto se ha desarrollado con el enfoque de una aplicación web para la gestión de clientes, por lo cual ha sido diseñada con una arquitectura modular, lo cual separa de manera efectiva los componentes frontend, Backend y base de datos. Con lo cual su propósito es permitir a los usuarios que puedan realizar operaciones CRUD, que viene siendo el crear, leer, actualizar y eliminar sobre un conjunto de datos de clientes.

1.2 Objetivos del proyecto

- Lograr diseñar una arquitectura escalabre y eficiente por medio de contenedores Docker.
- Implementar un backend en Node.js que se encargar de la gestión de la lógica del negocio y sus operaciones con la base de datos.
- Desarrollar un frontend interactivo que se pueda gestionar datos de clientes de una manera eficiente y sencilla para los usuarios que lo van a utilizar.
- Replicar la solución en la nube por medio de kubernetes para conseguir alta disponibilidad y escalabilidad.

2 Solución Local

2.1 Descripción de la Arquitectura Local

En la arquitectura local se ha implementado utilizar Docker Compose para orquestar los servicios backend y frontend, los cuales estos interactúan con una base de datos alojada en la nube de AWS, logrando encapsular cada uno en su propio contenedor. Asegurando que ambos componentes sean independientes y portátiles, provocando su facilitación en su despliegue y escalabilidad.

2.2 Diseño y separación de componentes

2.2.1 Backend

Por el lado del backend logramos encontrar que es implementado en Node.js y utiliza Express como framework, además de que se encuentra funcionalidades y configuraciones como:

- Manejo de operaciones CRUD para los datos de los clientes
- Se logra conectar a una base de datos PostgreSQL alojada en AWS.
- Publica una API REST por medio del puerto 3000 para que el frontend logre acceder a los datos
- Usa un archivo .env para el almacenamiento de credenciales y configuraciones sensibles, como por ejemplo el usuario, contraseña, host y puerto de la base de datos.
- Implementación de volúmenes para reflejar de manera automática los cambios en el código durante el desarrollo y se reflejen en el contenedor.

2.2.2 Base de datos

En la base de datos se haya que se utiliza PostgreSQL que está alojada en Amazon RDS, que es un sistema de gestión de base de datos relacional, que almacena toda la información de los clientes. Además de responder a las solicitudes de consulta y modificación realizadas por el backend y está configurada para la aceptación de conexiones desde el backend, utilizando credenciales seguras almacenadas en el archivo env.

2.2.3 Frontend

Por último, en el frontend se ha implementado en React, proporcionando la interfaz grafica en donde los usuarios interactúen con el sistema, logrando encontrar funciones claves y configuraciones como:

- Envío de solicitudes al backend para la realización de operaciones CRUD
- Se logra proporcionar formularios para agregar o editar clientes
- Mostrar datos de los clientes en una tabla interactiva
- Ha sido configurado en el puerto 8080
- Dependencia del backend para obtener datos y realizar operaciones.

2.3 Configuración de cada contenedor y como estos interactúan

```yaml
services:
  backend:
    build:
      context: ./crud-backend
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    env_file:
      - ./crud-backend/.env
    volumes:
      - ./crud-backend:/app


  frontend:
    build:
      context: ./crud-frontend
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    depends_on:
      - backend
```

Detalles de la configuración:

Backend:

- Se construye el contenedor del backend por medio del dockerfile ubicado en ./crud-backend
- Expone el servicio en el puerto 3000
- Carga credenciales y configuraciones para lograr la conexión a la base de datos en AWS desde un archivo .env
- Sincroniza código fuente local con su contenedor, logrando facilitar el desarrollo

Frontend:

- Se construye el contenedor del frontend por medio del dockerfile ubicado en ./crud-frontend
- Expone el servicio en el puerto 8080
- El depends_on se asegura que el backend esté operativo antes de iniciar el frontend

2.4 Interacción entre Componentes

En el flujo del trabajo, el usuario interactúa con el frontend (puerto 8080) para la gestión de clientes, después envía solicitudes HTTP al backend (puerto 3000), para que en el backend se procese las solicitudes y se conecte a la base de datos PostgreSQL en AWS para la realización de operaciones como almacenar, leer, actualizar y eliminación de datos. La base de datos responde al backend, que al mismo tiempo envía la respuesta al frontend.

3 Solución en la nube

Para la implementación en la nube, se utilizaron servicios de **Google Cloud** y **AWS** para desplegar y gestionar los diferentes componentes del sistema. Los contenedores del **frontend** y el **backend** fueron alojados en **Google Cloud Run**, un servicio totalmente administrado que permite implementar y ejecutar aplicaciones en contenedores basados en Docker de manera escalable y eficiente.

Por otro lado, la base de datos del sistema, implementada en **PostgreSQL**, fue alojada en **Amazon RDS (Relational Database Service)**. AWS RDS proporciona un entorno seguro, escalable y de alta disponibilidad para bases de datos relacionales, eliminando la necesidad de

realizar tareas complejas de administración como configuraciones de hardware, parches del sistema operativo o backups automáticos.

La arquitectura está configurada para que tanto el backend como el frontend puedan comunicarse de manera eficiente. Las URL y credenciales necesarias para la integración entre los servicios fueron gestionadas mediante variables de entorno, asegurando así una configuración flexible y segura en el entorno de despliegue.

4 Análisis y conclusiones

4.1 Análisis de rendimiento

En el ambiente local se encuentra ventajas evidentes como la baja latencia por medio de la proximidad de los componentes, es decir, que todos están corriendo en la misma máquina, además que hay facilidad de configuración y pruebas, perfecto para el desarrollo y depuración, Sin embargo, presenta problemas como la limitación de recursos en la maquina local, provocando que afecta la escalabilidad y si falla el sistema, no hay respaldo automático.

En el ambiente en la nube que se utilizan se analiza una escalabilidad automática gracias a las instancias del backend y el frontend de acuerdo la carga por medio de Google Cloud Run, también elimina la necesidad de gestionar manualmente infraestructura como clústeres o balanceadores de carga, así mismo las instancias desplegadas que se encuentra en Cloud Run están siempre disponibles mientras el tráfico persista. Sin embargo, se encuentran problemas en la latencia por culpa de la comunicación entre Cloud Run y AWS RDS que esto genera retrasos

en la consulta, y estos costos pueden aumentar con el trafico elevado por medio del modelo de pago por uso de Cloud Run y AWS RDS.

4.2 Latencia, escalabilidad y Disponibilidad

4.2.1 Latencia

En el ambiente local se logra encontrar que el promedio es muy bajo, aproximadamente 60ms debido a la proximidad de los servicios, a comparación de la nube que es mayor (entre 100-300ms) por factores como:

- Balanceo de tráfico en Cloud Run
- Comunicación entre Cloud Run y AWS RDS

4.2.2 Escalabilidad

Por medio local se encuentra limitada al hardware de la maquina local, donde no puede manejar un aumento significativo de tráfico, sin embargo, en la nube en términos de escalabilidad Cloud Run escala Horizontalmente por medio de creación de nuevas instancias según la demanda y AWS RDS escala verticalmente o mediante multi-AZ

4.2.3 Disponibilidad

Por medio local depende mucho de una maquina única, lo que hace susceptible a posibles fallos, por otra parte, en la nube manejado en este proyecto por Google Cloud garantiza alta disponibilidad de balanceos de cargas y diversas instancias. Además, que AWS RDS ofrece respaldo automático y replicación de diversas zonas

4.3 Retos y soluciones

En el desarrollo del proyecto se implementan diversas soluciones a problemas que se pueden evidenciar en el momento de utilizar la aplicación en la nube, tales como:

- Costos operativos: Cloud Run y AWS RDS presentan costos niveles elevados de costos de acuerdo al uso de esto, lo que puede generar un nivel alto de tráfico, una solución óptima es la configuración de conexiones que ofrece Google Cloud como Billing y aws cost explorer, consiguiendo limitar la escalabilidad en Cloud Run
- Seguridad en la conexión: aseguramiento de la conexión entre Cloud y AWS, una solución posible es configuración de conexiones como cifradas TLS y utilizar Google Secret Manager para el manejo de credenciales
- Latencia entre Cloud Run y AWS: el alojamiento de los servicios puede generar problemas a la hora del tiempo de respuesta; una solución factible seria la asignación de los servicios en regiones mas cercanas para la reducción de latencia y mejor tiempo de respuesta.

4.4 Reflexiones sobre el uso de tecnologías

En el recorrido del uso de tecnologías como AWS RDS se ha logrado evidenciar que es un buen servicio para la administración de bases de datos ya que esto maneja actualizaciones, respaldos y seguridad. Sin embargo, se requiere una gestión cuidadosa de las conexiones.

Por parte del Docker este proporciona gran portabilidad en el desarrollo local y Google Cloud Run ayuda con la eliminación de gestionar clústeres y servidores, dando así un escalado automático de los contenedores de acuerdo a la demanda.

4.5 Elementos claves

Escalabilidad: en el local no es escalable, sin embargo, por medio de la nube que se utiliza Cloud Run permite un medio de escalado horizontal, es decir, pude aumentar o disminuir de forma automática el número de instancias del servicio para sus variaciones de cargas del trabajo, y también AWS RDS que maneja su escalado vertical o multi AZ.

Fiabilidad: Por medio local es dependiente de una única máquina, a comparación de la nube que maneja balanceadores de carga y redundancia que maneja AWS RDS ayuda a tener una fiabilidad del sistema.

Costo: por el local no maneja ningún costo, la única limitante son los recursos que provee el hardware, por el otro lado, en la nube el costo es de acuerdo basado en el uso, donde puede incrementar de acuerdo a periodos de alto tráficos.

Seguridad: la única seguridad que maneja por medio local es la utilización del .env, provocando que sea una seguridad limitada por la maquina del anfitrión. Por otro lado, el uso de la nube maneja Google Secret Manager para la protección de sus credenciales y conexiones.

5 Conclusiones

Para concluir el proyecto ha cumplido con los objetivos del desarrollo de una página web modular para gestión de clientes, con lo cual se ha implementado en el entorno local y por la nube. Proporcionando en el local, el Docker Compose en la facilitación del despliegue y pruebas, sin embargo, con limitaciones de recursos y disponibilidad. Por otro lado, en el entorno de la nube como la implementación de la nube en Google Cloud Run y la debida gestión de la base de

datos en AWS RDS se destacan por su escalabilidad y nivel alto de disponibilidad, a pesar de presentar algunos desafíos como la latencia y los costos asociados.

Herramientas que han sido implementadas como el Docker por su portabilidad, el Cloud run por su automática escalabilidad y AWS RDS en la gestión de base de datos, se efectúan en diversos escenarios. En síntesis, el proyecto ha evidenciado una solución eficiente y escalable, mostrando el gran potencial de las tecnologías que han sido utilizadas y dejando unas bases en caso de querer hacer mejoras de rendimiento y adaptabilidad.