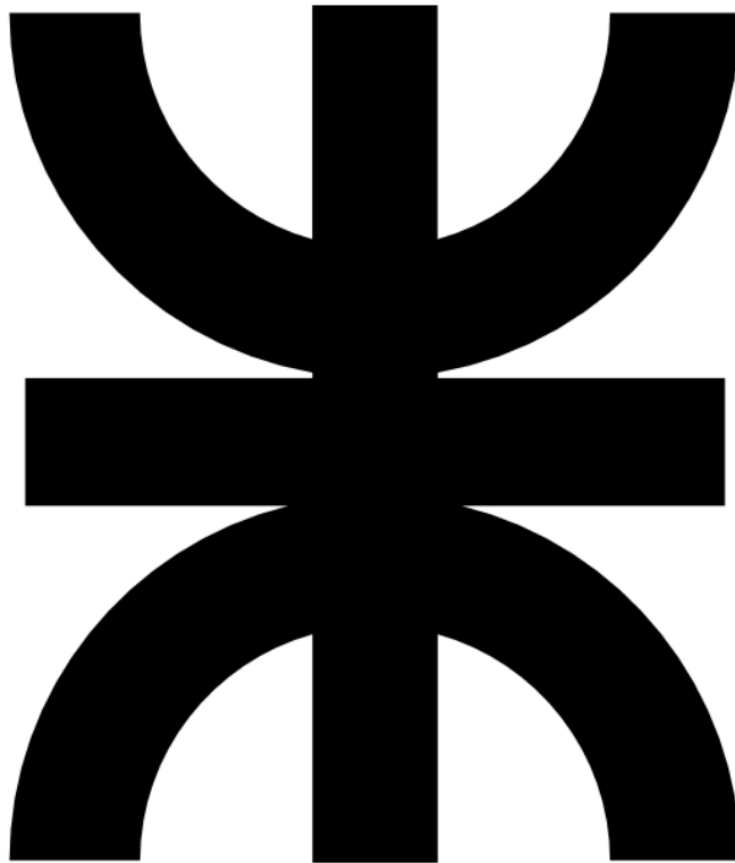

TRABAJO PRÁCTICO 1 - APLICACIÓN WEB Y SERVICIO REDIS CONTENERIZADOS



EQUIPO:

- FARIZANO CAÑETE, DIEGO NICOLÁS - 25445
 - MECOZZI, MATEO - 25606
-

PROFESOR:

- ING. JOSE ALEJANDRO FERNÁNDEZ

CATEDRA: DEVOPS

ISI - CICLO 2024 - SEGUNDO CUATRIMESTRE

RESISTENCIA, CHACO

1. Introducción

El presente informe detalla el desarrollo de una aplicación web para gestionar una lista "ToDo", utilizando un servicio de caché Redis. Ambas partes, la aplicación web y el servicio Redis, fueron contenerizadas y gestionadas con Docker Compose. El objetivo de este trabajo fue aplicar conocimientos de DevOps en la integración de servicios contenerizados, utilizando tecnologías modernas como Node.js, Redis y Docker.

2. Desarrollo

1. Selección de Tecnologías

- La aplicación fue desarrollada utilizando **Node.js** como entorno de ejecución, con **Express** como framework para el servidor web y **Redis** como sistema de caché. **Docker** y **Docker Compose** se utilizaron para contenerizar tanto la aplicación web como el servicio Redis, facilitando el manejo y la integración de los servicios.

2. Estructura del Proyecto

- El proyecto fue dividido en varios archivos clave:
 - **index.js**: Archivo principal que contiene la lógica del servidor, la gestión de las tareas y la conexión con Redis.
 - **docker-compose.yml**: Archivo de configuración que define y coordina los contenedores de la aplicación Node.js y Redis.
 - **index.html**: Archivo de interfaz de usuario, creado con HTML, para mostrar la lista de tareas y las acciones correspondientes.
 - **styles.css**: Archivo de estilos que define la presentación visual, mejorando la usabilidad de la aplicación.

3. Desarrollo de la Lógica de la Aplicación

- **Gestión de Tareas:**
 - Se desarrolló la funcionalidad para **agregar, eliminar, marcar y desmarcar tareas como completadas**, permitiendo al usuario gestionar sus tareas de manera intuitiva.
 - Las **tareas completadas** se mantienen en la lista original y se marcan visualmente como completadas utilizando un ícono y un estilo diferente. Además, el usuario tiene la opción de **desmarcar** una tarea si fue marcada como completada por error.
 - Las tareas se gestionan completamente dentro de Redis, donde cada tarea tiene un estado que determina si está **completada** o **pendiente**. El cliente interactúa con Redis mediante comandos **RPUSH**, **LREM**, y **LSET** para administrar las tareas y sus estados.

4. Interfaz de Usuario

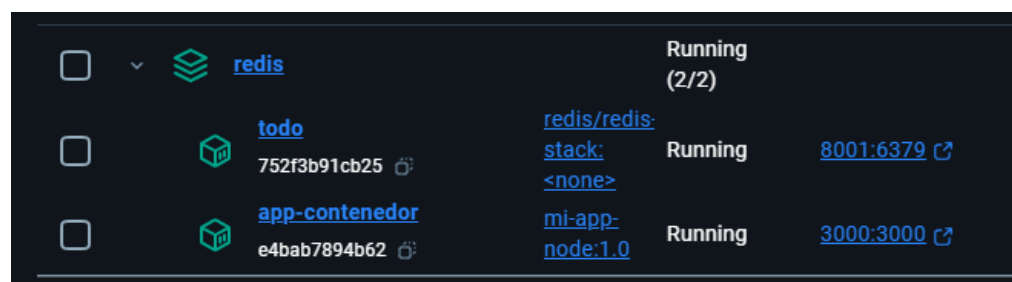
- La interfaz de usuario muestra las tareas pendientes y completadas en una misma lista. Las tareas completadas se muestran con un ícono visual y un estilo distinto para diferenciarlas de las tareas pendientes.
- Se añadieron botones de acción estilizados (íconos de **tilde** para completar y **basurero** para eliminar) al lado de cada tarea. Los botones están alineados a la derecha de cada tarea, utilizando **flexbox** para un diseño limpio.
- El usuario puede marcar una tarea como completada y, si lo desea, puede desmarcarla, devolviéndola a su estado de pendiente.

5. Contenerización con Docker

- Para facilitar el despliegue y la ejecución, se creó un archivo **Dockerfile** que define cómo construir la imagen del contenedor de la aplicación Node.js. Redis también fue contenerizado utilizando la imagen oficial `redis/redis-stack` disponible en Docker Hub.
- **Docker Compose** se utilizó para coordinar ambos contenedores (la aplicación web y Redis), definiendo las dependencias entre ellos y asegurando que la aplicación pueda conectarse correctamente al servicio Redis.

Contenedor compuesto de:

- **todo:** Server Redis
- **server-1:** App



<input type="checkbox"/>		redis		Running (2/2)
<input type="checkbox"/>		todo 752f3b91cb25	redis/redis-stack: <none>	Running 8001:6379
<input type="checkbox"/>		app-contenedor e4bab7894b62	mi-app-node:1.0	Running 3000:3000

Imágenes utilizadas:



<input checked="" type="checkbox"/>	redis/redis-stack 667da61fde3d	latest	In use	2 months ago	779.62 MB
<input checked="" type="checkbox"/>	mi-app-node 215a8514c580	1.0	In use	2 hours ago	136.51 MB

6. Configuración del archivo **docker-compose.yml**:

- La aplicación Node.js y el servicio Redis se ejecutan en redes internas para que puedan comunicarse entre sí.
- El archivo de configuración **docker-compose.yml** especifica los puertos en los que los contenedores estarán disponibles, así como la conexión entre ellos (Redis se conecta a la aplicación web a través de la red interna de Docker).

7. Estilización y Usabilidad

- Se añadió CSS para mejorar la presentación visual de la aplicación. Los botones de **completar** y **eliminar** fueron reemplazados por íconos (un tilde para completar y un basurero para eliminar), lo que hizo la interfaz más intuitiva.
- Se utilizó **Flexbox** para organizar los elementos de la lista de tareas, asegurando que los íconos de acción estuvieran alineados a la derecha de cada tarea, proporcionando una interfaz limpia y clara.

8. Gestión de Tareas Completadas

- En lugar de mover las tareas completadas a una lista separada, las tareas se mantienen en la misma lista en Redis, pero con un estado visualmente distinto en la interfaz (tachado). Esto permite al usuario ver todas sus tareas, tanto completadas como pendientes, en un solo lugar.
- Se agregó la opción de **desmarcar** tareas completadas, lo que permite al usuario corregir cualquier error en caso de que haya marcado una tarea como completada por accidente.



Pruebas y Verificación

- Durante el desarrollo, se realizaron pruebas para garantizar que las tareas se agregaran y manipularan correctamente en Redis. Esto incluyó la validación de que las tareas pudieran agregarse, completarse, eliminarse, y desmarcarse sin problemas.
- Se utilizaron logs y consultas directas a Redis para verificar que los datos estaban siendo almacenados correctamente y que las operaciones de marcado y desmarcado funcionaban como se esperaba.

```
mateo@DESKTOP-A75UKB0 MINGW64 ~/OneDrive/2 - Educación/5to AÑO/DevOps/redis (main)
$ docker exec -it redis-todo redis-cli
127.0.0.1:6379> LRange todos 0 -1
1) "Armar la app web"
2) "\xe2\x9c\x85 Levantar redis"
3) "Componer imagenes"
4) "Hola"
127.0.0.1:6379> |
```

3. Resultados Obtenidos

1. Aplicación Web (ToDo List):

- Se desarrolló una aplicación web sencilla que permite gestionar una lista de tareas, donde se pueden **agregar, marcar como completadas y eliminar** tareas.
- Las tareas completadas no se eliminan sino que se muestran al usuario como completadas. Además el usuario puede desmarcarlas en caso de que se haya equivocado
- Se añadieron mejoras de usabilidad, incluyendo un diseño limpio y sencillo con botones de **completar** y **eliminar** estilizados como un **tilde (✓)** y un **basurero (🗑️)**.

2. Contenerización:

- Se contenerizó tanto la aplicación web como el servicio Redis utilizando Docker.
- La interacción entre los servicios fue gestionada correctamente a través de **Docker Compose**, facilitando la administración y despliegue de los contenedores. La app y el server de redis tienen imágenes distintas pero se levantan gracias a docker compose en un contenedor que aloja dos contenedores uno por cada imagen.

3. Integración con Redis:

- Redis fue utilizado como una caché para almacenar y recuperar la lista de tareas. Se integró con éxito con la aplicación Node.js, permitiendo realizar operaciones como agregar, completar y eliminar tareas.

4. Dificultades Encontradas

1. Configuración Inicial de Docker:

- Al principio, se encontraron dificultades con la configuración de las redes de Docker y la conexión entre la aplicación y Redis. Sin embargo, estos problemas fueron resueltos ajustando los parámetros de `docker-compose.yml` y utilizando las dependencias de red adecuadas.

- Luego se podía acceder desde el navegador a la app, lo que se solucionó creando el dockerfile en la carpeta raíz del proyecto.
2. **Errores HTTP 500:**
 - Durante el desarrollo, se encontraron errores de tipo 500 (Internal Server Error), relacionados con problemas de conexión con Redis o de renderizado de la página. Estos errores fueron depurados mediante la adición de logs y manejo de excepciones.
 3. **Alineación y Diseño de la Interfaz:**
 - A la hora de estilizar la aplicación, hubo algunos retos para alinear correctamente los botones de acción de cada tarea. Esto se solucionó utilizando **flexbox** en el CSS para gestionar mejor el layout.
-

5. Posibles Mejoras

1. **Autenticación y Manejo de Sesiones:**
 - Se podría agregar una capa de autenticación para que cada usuario tenga su propia lista de tareas. Esto también incluiría la posibilidad de manejar sesiones con Redis para mejorar la experiencia del usuario.
2. **Persistencia de Datos:**
 - Actualmente, las tareas se almacenan en Redis, que es un sistema de caché en memoria. Una posible mejora sería agregar persistencia en una base de datos como **MongoDB** o **PostgreSQL** para asegurar que los datos no se pierdan cuando Redis se reinicie.
3. **Notificaciones y Recordatorios:**
 - Se podría implementar un sistema de recordatorios o notificaciones para las tareas. Por ejemplo, enviar correos electrónicos o notificaciones push cuando las tareas estén próximas a vencerse.
4. **Mejoras en la Interfaz:**
 - La interfaz de usuario puede ser más interactiva y moderna utilizando frameworks de frontend como **React** o **Vue.js** para mejorar la experiencia del usuario.
5. **Despliegue en Producción:**
 - Si la aplicación se despliega en un entorno de producción, sería recomendable configurar un servicio de **CI/CD** (como GitHub Actions o Jenkins) para automatizar el despliegue, así como implementar un sistema de monitoreo para los contenedores (por ejemplo, con **Prometheus** y **Grafana**).

6. Conclusión

El trabajo fue exitoso, logrando contenerizar una aplicación web y un servicio Redis que se integran de manera eficiente. Se resolvieron las dificultades encontradas y se identificaron mejoras potenciales para el futuro que permitirían escalar la solución y mejorar la experiencia del usuario. Este ejercicio proporcionó una experiencia valiosa en el manejo de aplicaciones contenerizadas y la integración de servicios modernos en entornos DevOps.