Evaluación Docker y Comunicación

En este documento se describe de forma breve el funcionamiento de la arquitectura implementada bajo una topología de anillo, utilizada para cumplir con el requerimiento de que tres nodos (A, B y C) procesen y transformen un mensaje de manera secuencial hasta cerrar el ciclo de regreso al nodo inicial.

La elección de esta arquitectura se debe a que garantiza un flujo circular y controlado, permitiendo que cada nodo cumpla una función específica dentro del proceso. Aunque su estructura aparenta ser sencilla, puede volverse compleja al considerar las reglas estrictas de transformación y la comunicación entre contenedores mediante WebSockets.

En la implementación realizada:

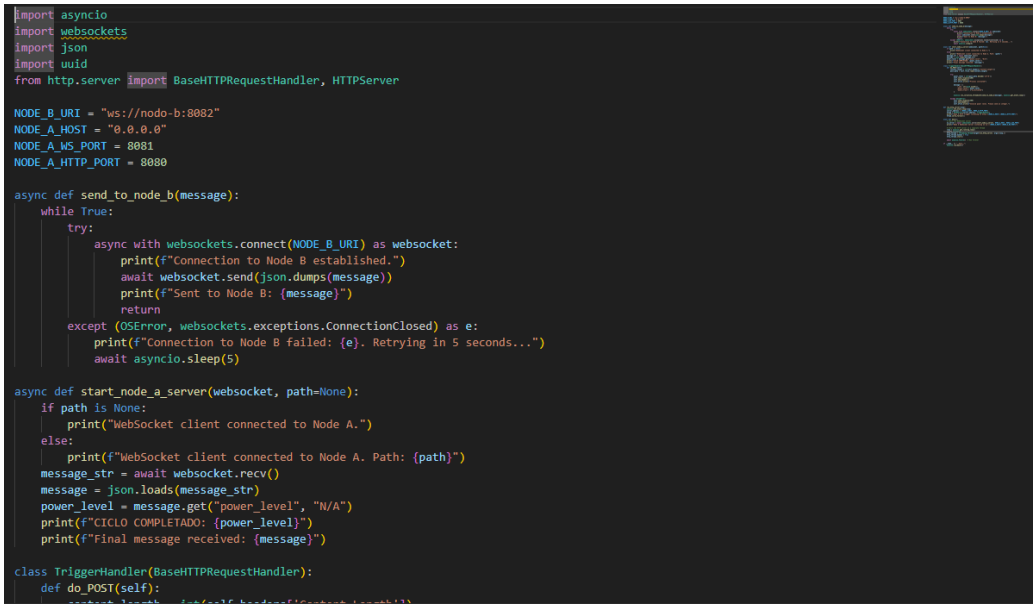
- Nodo A inicia el proceso, recibe el valor del usuario, genera el JSON y lo envía al Nodo B.
- Nodo B aplica la lógica de transformación según si el power_level es par o impar y envía el mensaje al Nodo C.
- Nodo C realiza la verificación final, modifica el power_level y cierra el ciclo devolviendo el mensaje a A.

Cuando el Nodo A recibe nuevamente el mensaje, imprime en consola:
“CICLO COMPLETADO: [valor final]”.

La arquitectura se despliega completamente con docker-compose, utilizando los alias de servicio para la comunicación y al menos un Dockerfile personalizado. Este diseño permite visualizar cómo los tres contenedores trabajan de manera distribuida y coordinada, cumpliendo el ciclo completo de procesamiento solicitado.

```
PS C:\Users\mlata\OneDrive\Escritorio\wqsadsa\Evaluaci-n-Docker-y-Comunicaci-n_Michael_Lata> docker-compose logs -f
nodo-c-1  | Node C WebSocket Server listening on ws://0.0.0.0:8083
nodo-b-1  | Node B WebSocket Server listening on ws://0.0.0.0:8082
nodo-a-1  | Node A WebSocket Server listening on ws://0.0.0.0:8081
nodo-a-1  | Node A HTTP Trigger listening on http://0.0.0.0:8080
nodo-a-1  | 172.20.0.1 - - [27/Nov/2025 02:18:49] "POST / HTTP/1.1" 200 -
nodo-b-1  | Node A connected to Node B.
nodo-b-1  | Received from Node A: {'_id': 'faecd30c-ecdc-483d-ae57-83e519288043', 'power_level': 50, 'audit_trail': ['A_initiated']}
nodo-a-1  | Connection to Node B established.
nodo-a-1  | Sent to Node B: {'_id': 'faecd30c-ecdc-483d-ae57-83e519288043', 'power_level': 50, 'audit_trail': ['A_initiated']}
nodo-c-1  | Node B connected to Node C.
nodo-c-1  | Received from Node B: {'_id': 'faecd30c-ecdc-483d-ae57-83e519288043', 'power_level': 100, 'audit_trail': ['A_initiated', 'B_processed']}
nodo-b-1  | Connection to Node C established.
nodo-b-1  | Sent to Node C: {'_id': 'faecd30c-ecdc-483d-ae57-83e519288043', 'power_level': 100, 'audit_trail': ['A_initiated', 'B_processed']}
nodo-a-1  | WebSocket client connected to Node A.
nodo-a-1  | CICLO COMPLETADO: 95
nodo-a-1  | Final message received: {'_id': 'faecd30c-ecdc-483d-ae57-83e519288043', 'power_level': 95, 'audit_trail': ['A_initiated', 'B_processed', 'C_verified']}
nodo-c-1  | Connection to Node A established.
nodo-c-1  | Sent to Node A: {'_id': 'faecd30c-ecdc-483d-ae57-83e519288043', 'power_level': 95, 'audit_trail': ['A_initiated', 'B_processed', 'C_verified']}
```

Por otro lado, los nodos resultaron funcionales y mantienen una estructura muy similar entre sí; las diferencias principales se encuentran únicamente en las funciones específicas que ejecuta cada uno dentro del ciclo.



```python
import asyncio
import websockets
import json
import uuid
from http.server import BaseHTTPRequestHandler, HTTPServer

NODE_B_URI = "ws://nodo-b:8082"
NODE_A_HOST = "0.0.0.0"
NODE_A_WS_PORT = 8081
NODE_A_HTTP_PORT = 8080

async def send_to_node_b(message):
    while True:
        try:
            async with websockets.connect(NODE_B_URI) as websocket:
                print(f"Connection to Node B established.")
                await websocket.send(json.dumps(message))
                print(f"Sent to Node B: {message}")
                return
        except (OSError, websockets.exceptions.ConnectionClosed) as e:
            print(f"Connection to Node B failed: {e}. Retrying in 5 seconds...")
            await asyncio.sleep(5)

async def start_node_a_server(websocket, path=None):
    if path is None:
        print("WebSocket client connected to Node A.")
    else:
        print(f"WebSocket client connected to Node A. Path: {path}")
    message_str = await websocket.recv()
    message = json.loads(message_str)
    power_level = message.get("power_level", "N/A")
    print(f"CICLO COMPLETADO: {power_level}")
    print(f"Final message received: {message}")

class TriggerHandler(BaseHTTPRequestHandler):
    def do_POST(self):
```

En este proyecto fue necesario realizar tres commits, principalmente debido a errores de sincronización que surgieron durante la implementación y comunicación entre los nodos. Cada commit permitió corregir y estabilizar el flujo del mensaje dentro de la topología en anillo.

Por otro lado, también se trabajó en el archivo docker-compose.yaml, el cual cumple un rol fundamental en la orquestación de los tres contenedores. Gracias a este archivo se definieron los servicios, los alias de red y la forma en que los nodos se comunican entre sí, permitiendo levantar toda la arquitectura con un solo comando.

```yaml
ocker-compose.yml
  Run All Services
services:
  Run Service
  nodo-a:
    build: .
    command: sh -c "rm -rf __pycache__ && /usr/local/bin/python -u The_Initiator.py"
    ports:
      - "8080:8080"
    networks:
      - app-network

  Run Service
  nodo-b:
    build: .
    command: sh -c "rm -rf __pycache__ && /usr/local/bin/python -u The_Transformer.py"
    networks:
      - app-network

  Run Service
  nodo-c:
    build: .
    command: sh -c "rm -rf __pycache__ && /usr/local/bin/python -u The_Auditor.py"
    networks:
      - app-network

networks:
  app-network:
    driver: bridge
```