Prueba 1_ Sistemas Distribuidos

Nombre: Michael Franco

Repositorio: https://github.com/michfranko/Prueba1Sd_FrancoM.git

NODO A:

1. Servidor HTTP para el disparo inicial externo

```
const serverHttp = http.createServer((req, res) => {
    if (req.url.startsWith('/trigger') && req.method === 'POST') {
        let body = '';
        req.on('data', chunk => { body += chunk; });
        req.on('end', () => {
            try {
                const data = JSON.parse(body);
                const initialPower = data.initial_power_value;

                if (typeof initialPower !== 'number') {
                    res.writeHead(400, { 'Content-Type': 'text/plain' });
                    res.end('Se esperaba un valor numerico en el body: {"initial_power_value": N}');
                    return;
                }

                // Generar UUID simple para el _id
                const message = {
                    _id: `uuid-${Date.now()}-${Math.random().toString(36).substr(2, 9)}`,
                    power_level: initialPower,
                    audit_trail: []
                };

                console.log(`[A] Iniciando ciclo con power_level: ${initialPower}`);
                // Enviar al Nodo B via WS
                sendToB(message);

                res.writeHead(200, { 'Content-Type': 'text/plain' });
                res.end(`Mensaje ${message._id} iniciado y enviado a Nodo B.`);

            } catch (e) {
                res.writeHead(400, { 'Content-Type': 'text/plain' });
                res.end('Error parsing request body.');
            }
        });
    } else {
        res.writeHead(404);
        res.end();
    }
});

serverHttp.listen(PORT_HTTP, () => {
    console.log(`[A] Servidor HTTP escuchando en el puerto ${PORT_HTTP} (Externo) para triggers.`);
});
```

Función para enviar al Nodo B

```
function sendToB(message) {
    const ws = new WebSocket(`ws://${UPSTREAM_HOST}:${UPSTREAM_PORT}`);

    ws.onopen = () => {
        console.log(`[A] Conectado a B, enviando mensaje ${message._id}`);
        ws.send(JSON.stringify(message));
    };

    ws.onerror = (err) => {
        console.error(`[A] Error de conexión a B: ${err.message}`);
    };

    ws.onclose = () => {
        // La conexión a B es efímera, se cierra después de enviar.
    };
}
```

2. **Servidor WebSocket para recibir del Nodo C (cierre del anillo)**

```
const wss = new WebSocket.Server({ port: PORT_WS_INTERNAL });

wss.on('connection', ws => {
    ws_c_connection = ws;
    console.log('[A] Conexión WS entrante desde Nodo C establecida.');

    ws.on('message', message => {
        const jsonMessage = JSON.parse(message);
        // Lógica de cierre y validación final
        console.log(`\n--- CICLO COMPLETADO: ${jsonMessage.power_level} ---`);
        console.log(`ID: ${jsonMessage._id}, Auditoría: ${jsonMessage.audit_trail.join(' -> ')}\n`);
        ws.close(); // Cerrar la conexión después de procesar el ciclo.
    });

    ws.on('close', () => {
        ws_c_connection = null;
    });
});

console.log(`[A] Servidor WS (para recibir de C) escuchando internamente en el puerto ${PORT_WS_INTERNAL}`);
```

NODO B:

Servidor WebSocket para recibir de A

```
const wss = new WebSocket.Server({ port: PORT_WS });

wss.on('connection', ws => {
    console.log('[B] Conexión WS entrante (desde A) establecida.');

    ws.on('message', message => {
        const jsonMessage = JSON.parse(message);
        console.log(`[B] Recibido: ${JSON.stringify(jsonMessage)}`);

        // Lógica de Negocio Paso 2
        let { power_level, audit_trail } = jsonMessage;

        if (power_level % 2 === 0) {
            jsonMessage.power_level = power_level * 2; // PAR: Multiplica por 2
        } else {
            jsonMessage.power_level = power_level + 1; // IMPAR: Suma 1
        }

        jsonMessage.audit_trail.push("B_processed");

        console.log(`[B] Modificado a power_level: ${jsonMessage.power_level}`);

        // Enviar al Nodo C
        sendToC(jsonMessage);
    });
});
```

Función para enviar al Nodo C (se conecta efímeramente)

```
function sendToC(message) {
    const ws = new WebSocket(`ws://${UPSTREAM_HOST}:${UPSTREAM_PORT}`);

    ws.onopen = () => {
        console.log(`[B] Conectado a C, enviando mensaje ${message._id}`);
        ws.send(JSON.stringify(message));
    };

    ws.onerror = (err) => {
        console.error(`[B] Error de conexión a C: ${err.message}`);
    };
}
```

NODO C:

Servidor WebSocket para recibir de B

```
const wss = new WebSocket.Server({ port: PORT_WS });

wss.on('connection', ws => {
    console.log('[C] Conexión WS entrante (desde B) establecida.');

    ws.on('message', message => {
        const jsonMessage = JSON.parse(message);
        console.log(`[C] Recibido: ${JSON.stringify(jsonMessage)}`);

        // Lógica de Negocio Paso 3
        jsonMessage.power_level -= 5; // Resta 5
        jsonMessage.audit_trail.push("C_verified");

        console.log(`[C] Modificado a power_level: ${jsonMessage.power_level}`);

        // Enviar de vuelta al Nodo A
        sendToA(jsonMessage);
    });
});

console.log(`[C] Servidor WS escuchando internamente en el puerto ${PORT_WS}`);
```

Función para enviar al Nodo A (cierre del anillo)

```
function sendToA(message) {
    const ws = new WebSocket(`ws://${UPSTREAM_HOST}:${UPSTREAM_PORT}`);

    ws.onopen = () => {
        console.log(`[C] Conectado a A (cerrando anillo), enviando mensaje ${message._id}`);
        ws.send(JSON.stringify(message));
        ws.close();
    };

    ws.onerror = (err) => {
        console.error(`[C] Error de conexión a A: ${err.message}`);
    };
}
```

DOCKER COMPOSE:

```yaml
services:
  ▷Run Service
  node-a:
    build:
      context: node_a
      dockerfile: Dockerfile
    ports:
      - "8080:8080" # Expone el puerto 8080 al host para el disparo inicial
    networks:
      security-net:
        aliases:
          - node-a

  ▷Run Service
  node-b:
    build:
      context: node_b
      dockerfile: Dockerfile
    networks:
      security-net:
        aliases:
          - node-b
    # No expone puertos al host

  ▷Run Service
  node-c:
    build:
      context: node_c
      dockerfile: Dockerfile
    networks:
      security-net:
        aliases:
          - node-c
    # No expone puertos al host

networks:
  security-net:
    driver: bridge
```

DOCKERFILE:

```
# Usar una imagen base oficial de Node.js
FROM node:18-alpine

# Crear directorio de trabajo
WORKDIR /usr/src/app


# Instalar dependencias necesarias para 'ws' en Alpine
RUN apk add --no-cache python3 make g++

# Instalar la librería 'ws' globalmente o localmente.
# Para este ejemplo simple, instalamos localmente después de copiar app.js
COPY app.js ./
RUN npm install ws

# Comando para ejecutar la aplicación cuando el contenedor se inicie
CMD ["node", "app.js"]
```

ERRORES:

```
PS C:\Users\Estudiante\Desktop\Prueba1_FrancoM> docker-compose up
Compose can now delegate builds to bake for better performance.
 To do so, set COMPOSE_BAKE=true.
[+] Building 0.0s (0/0)
unable to prepare context: path "C:\\Users\\Estudiante\\Desktop\\Prueba1_FrancoM\\node_b" not found
```

- Esto ocurre si el directorio ha sido movido, renombrado, eliminado o si hay un error de escritura en la ruta especificada en el archivo YAML. También puede ser un problema de permisos o de sincronización si se usa WSL (Windows Subsystem for Linux).