Conectando seu App ao Mundo do IoT: Um Guia Prático para o Gateway do LSDI

Olá, desenvolvedor! Se você está aqui, provavelmente quer conectar sua aplicação (seja um site, um app mobile ou qualquer outro front-end) aos dados que fluem pelo broker do Laboratório de Sistemas Distribuídos e Inteligentes (LSDI). A boa notícia é que você está no lugar certo. Este guia foi feito para te ajudar a fazer exatamente isso, de uma forma simples e direta.

Para que serve este guia?

Imagine que o LSDI tem uma fonte inesgotável de dados sendo transmitidos em tempo real, como informações de sensores, dispositivos e outros sistemas. Esses dados são publicados usando um protocolo chamado **MQTT**, que é super eficiente para Internet das Coisas (IoT), mas não é diretamente compatível com os navegadores de internet.

É aí que entra o **WebSocket**, uma tecnologia que permite uma comunicação em tempo real entre seu aplicativo e um servidor. A solução que vamos construir aqui é uma **ponte (ou *gateway*)** que "traduz" os dados do formato MQTT do LSDI para o formato WebSocket que seu aplicativo entende.

Neste guia, vamos montar, passo a passo, um ambiente na sua própria máquina de desenvolvimento que faz o seguinte:

- **Se conecta ao broker do LSDI** para "ouvir" todos os dados que estão sendo transmitidos.
- **Cria um "espelho" desses dados** em um broker local na sua máquina.
- **Disponibiliza esses dados via WebSocket**, de forma que seu aplicativo possa se inscrever e recebê-los em tempo real.
- **Garante que sua aplicação seja "somente leitura"**, ou seja, ela pode receber dados, mas não pode enviar nada de volta para o LSDI, evitando qualquer interferência acidental.

Vamos usar ferramentas como **Docker** e **Mosquitto**, mas não se preocupe se você não for um especialista nelas. Vamos explicar tudo o que você precisa saber.

Entendendo as Peças do Quebra-Cabeça

Antes de colocarmos a mão na massa, vamos entender rapidamente as tecnologias que vamos usar. Conhecê-las vai tornar o processo todo muito mais claro.

O que é MQTT?

MQTT (Message Queuing Telemetry Transport) é um protocolo de mensagens leve, ideal para conectar dispositivos com recursos limitados (como sensores) ou em redes com baixa largura de banda. Ele funciona em um modelo de **publicação/assinatura (publish/subscribe)**:

- **Publisher (Publicador):** Um dispositivo ou serviço que envia mensagens para um "tópico" específico (pense em um tópico como um canal de comunicação, por exemplo, `sensor/temperatura/sala1`).
- **Subscriber (Assinante):** Uma aplicação que se "inscreve" em um ou mais tópicos para receber as mensagens enviadas para eles.
- **Broker:** O coração do sistema. É um servidor que recebe as mensagens dos publicadores e as entrega para os assinantes corretos. O LSDI tem um broker público que vamos acessar.

O que é WebSocket?

WebSocket é um protocolo de comunicação que permite uma conexão interativa e de mão dupla entre o navegador do usuário e um servidor. Diferente do modelo tradicional de requisição-resposta do HTTP, onde o cliente sempre tem que iniciar a comunicação, o WebSocket mantém um canal aberto, permitindo que o servidor envie dados para o cliente a qualquer momento. É perfeito para aplicações que precisam de atualizações em tempo real, como chats, jogos online e, no nosso caso, visualização de dados de IoT.

Por que precisamos de uma "ponte" (Gateway)?

Navegadores web não "falam" MQTT nativamente. Eles "falam" HTTP e WebSocket. Nosso gateway, portanto, atua como um tradutor. Ele se conecta ao broker MQTT do LSDI, recebe os dados e os retransmite por uma conexão WebSocket que seu aplicativo web pode consumir facilmente.

O que são Docker e Mosquitto?

- **Docker:** É uma plataforma que permite "empacotar" uma aplicação com todas as suas dependências em uma unidade chamada **contêiner**. Pense nisso como uma caixa que tem tudo o que o nosso gateway precisa para funcionar, garantindo que ele rode da mesma forma em qualquer máquina. Isso simplifica enormemente a instalação e a configuração.
- **Mosquitto:** É uma implementação de código aberto de um broker MQTT. Nós vamos usá-lo para criar nosso broker local, que receberá os dados do LSDI e os servirá via WebSocket para a sua aplicação.

Mão na Massa: Configurando seu Ambiente

Agora que entendemos a teoria, vamos para a prática. A configuração é bem simples e se resume a criar alguns arquivos de texto e rodar um único comando.

1. Estrutura de Pastas

Primeiro, crie uma pasta principal para o nosso projeto. Você pode chamá-la de `gateway-mqtt-ws`. Dentro dela, crie a seguinte estrutura de pastas e arquivos (você pode criar os arquivos vazios por enquanto, vamos preenchê-los a seguir):

Plain Text

```
gateway-mqtt-ws/
├── docker-compose.yml
└── config/
    ├── mosquitto.conf
    └── acl
```

- `docker-compose.yml`: Este arquivo vai dizer ao Docker como construir e rodar nosso gateway.
- `config/mosquitto.conf`: O arquivo de configuração do nosso broker Mosquitto local.
- `config/acl`: Um arquivo para definir as regras de acesso (ACL - Access Control List), garantindo que sua aplicação só possa ler os dados.

2. Configurando o Docker (`docker-compose.yml`)

Abra o arquivo `docker-compose.yml` e cole o seguinte conteúdo:

YAML

```yaml
services:
  mosquitto-ws-gateway:
    image: eclipse-mosquitto:2
    container_name: mosquitto-ws-gateway
    ports:
      - "8083:8083"            # Porta WebSocket que seu app vai usar
    volumes:
      - ./config/mosquitto.conf:/mosquitto/config/mosquitto.conf:ro
      - ./config/acl:/mosquitto/config/acl:ro
      - ./data:/mosquitto/data
      - ./log:/mosquitto/log
    restart: unless-stopped
```

O que este arquivo faz?

- `image: eclipse-mosquitto:2` : Diz ao Docker para usar a imagem oficial do Mosquitto.
- `ports: - "8083:8083"` : Expõe a porta 8083 do contêiner para a porta 8083 da sua máquina. É nesta porta que seu aplicativo vai se conectar.
- `volumes` : Mapeia os arquivos de configuração que criamos (`mosquitto.conf` e `acl`) para dentro do contêiner, para que o Mosquitto possa usá-los.

3. Configurando o Broker Mosquitto (`config/mosquitto.conf`)

Este é o arquivo mais importante, onde a mágica acontece. Cole o seguinte conteúdo nele:

Plain Text

```
# ====== BROKER LOCAL (O que seu app vai usar) ======
persistence true
persistence_location /mosquitto/data/
log_dest stderr
allow_anonymous true                # Permite conexões sem login/senha
(ideal para dev)
acl_file /mosquitto/config/acl      # Aponta para nosso arquivo de regras de
acesso

# Listener WebSocket para os clientes (seu app)
listener 8083
protocol websockets

# ====== PONTE PARA O BROKER DO LSDI (Somente Leitura) ======
# Esta seção configura a conexão com o broker do LSDI
connection lsdi-in
address lsdi.ufma.br:1883           # Endereço do broker do LSDI
bridge_protocol_version mqttv311
cleansession true
try_private false
keepalive_interval 60

# A linha mais importante: "in" significa que só vamos RECEBER dados do LSDI
# O "#" significa que queremos receber de TODOS os tópicos.
topic # in 0

# Se o LSDI precisar de autenticação, descomente e preencha as linhas abaixo:
# remote_username SEU_USUARIO
# remote_password SUA_SENHA
```

O que este arquivo faz?

- **Broker Local:** Configura o broker que vai rodar na sua máquina, habilitando a porta 8083 para conexões WebSocket e permitindo que clientes se conectem anonimamente (para facilitar o desenvolvimento).
- **Ponte para o LSDI:** A seção `connection lsdi-in` estabelece a ponte. A diretiva `topic # in 0` é a chave aqui: ela diz ao nosso broker para se inscrever em todos os tópicos (`#`) do broker do LSDI e trazer as mensagens para dentro (`in`), mas **não** enviar nada na direção contrária (`out`).

4. Definindo as Regras de Acesso (`config/acl`)

Para garantir que sua aplicação não possa, acidentalmente, publicar mensagens, vamos criar uma regra bem simples. Cole o seguinte no arquivo `acl` :

Plain Text

```
# Permite que qualquer cliente leia (se inscreva) em qualquer tópico.
pattern read #

# Como não há nenhuma regra "write", qualquer tentativa de publicar será
bloqueada.
```

Com isso, qualquer cliente que se conectar ao seu broker local só terá permissão de leitura.

5. Subindo o Gateway

Agora que todos os arquivos estão prontos, abra um terminal na pasta raiz do seu projeto (`gateway-mqtt-ws/`) e execute o seguinte comando:

Bash

```
docker compose up -d
```

O Docker vai baixar a imagem do Mosquitto (se for a primeira vez) e iniciar seu gateway em segundo plano (`-d`). Se tudo deu certo, você já tem uma ponte funcionando!

Testando sua Conexão

Vamos verificar se tudo está funcionando como esperado.

Teste 1: Verificando a Conexão WebSocket

Você pode usar uma ferramenta como o `curl` para simular um "aperto de mão" WebSocket. Se você estiver no Windows (usando PowerShell), pode rodar:

Plain Text

```
curl.exe -i --http1.1 -H "Connection: Upgrade" -H "Upgrade: websocket" -H
"Sec-WebSocket-Version: 13" -H "Sec-WebSocket-Key: SGVsbG8sIQ==""
http://localhost:8083/
```

A resposta deve conter `HTTP/1.1 101 Switching Protocols`, o que indica que o servidor WebSocket está no ar e pronto para aceitar conexões.

Teste 2: Recebendo Mensagens

Para um teste mais completo, você pode usar uma ferramenta de linha de comando para MQTT como o `mqttx` (que você pode instalar via `npm install -g mqttx`).

1. **Inscreva-se para receber mensagens do seu gateway local:**

- `-t '#'`: Inscreve-se em todos os tópicos.
- `-h localhost -p 8083`: Conecta no seu gateway local na porta 8083.
- `-l ws`: Especifica que a conexão é via WebSocket.

1. **Tente publicar uma mensagem (e veja falhar):**

Integrando com seu Front-End

Agora a parte mais legal: conectar sua aplicação. Aqui está um exemplo simples de como fazer isso em uma página HTML com JavaScript, usando a popular biblioteca `mqtt.js`.

HTML

```
<!DOCTYPE html>
<html>
<head>
    <title>Meu App com LSDI</title>
    <script src="https://unpkg.com/mqtt/dist/mqtt.min.js"></script>
</head>
<body>
    <h1>Dados do LSDI em Tempo Real</h1>
    <div id="messages"></div>

    <script>
        const client = mqtt.connect('ws://localhost:8083', {
            clientId: 'web_' + Math.random().toString(16).slice(2)
        });

        client.on('connect', () => {
            console.log('Conectado ao gateway WebSocket!');
            // Inscreva-se nos tópicos que te interessam
```

```
            // Cuidado ao usar '#' em produção, pode gerar muito tráfego!
            client.subscribe('#', (err) => {
                if (!err) {
                    console.log('Inscrito com sucesso!');
                }
            });
        });

        client.on('message', (topic, payload) => {
            console.log('Mensagem recebida:', topic, payload.toString());
            const messagesDiv = document.getElementById('messages');
            const newMessage = document.createElement('p');
            newMessage.textContent = `Tópico: ${topic} - Mensagem:
${payload.toString()}`;
            messagesDiv.appendChild(newMessage);
        });

        client.on('error', (err) => {
            console.error('Erro de conexão:', err);
            client.end();
        });
    </script>
</body>
</html>
```

Como usar este código:

1. Salve este código como um arquivo `index.html`.
2. Abra este arquivo no seu navegador.
3. Abra o console do desenvolvedor (geralmente com a tecla F12) para ver as mensagens de log.
4. As mensagens recebidas do LSDI aparecerão tanto no console quanto na página.

Resumo

Parabéns! Você construiu um gateway robusto e seguro que permite que sua aplicação web consuma dados em tempo real do broker do LSDI. Recapitulando:

- Usamos o **Docker** para criar um ambiente isolado e fácil de gerenciar.
- Configuramos o **Mosquitto** para atuar como uma ponte (`bridge`) que busca dados do LSDI.
- Disponibilizamos esses dados via **WebSocket** na porta 8083.
- Garantimos que a comunicação é **somente leitura**, protegendo o broker do LSDI.

Agora você tem uma base sólida para começar a construir aplicações incríveis com os dados do LSDI. Boas codificações!