MQTT Project Presentation

Overview

This project demonstrates the use of MQTT (Message Queuing Telemetry Transport) for real-time communication between a **Mosquitto broker** hosted on an Azure server and a **React application client** running on another Azure server. Both the broker and the client are deployed using Docker Compose for simplicity and reproductability.

Key Components:

1. **Mosquitto Broker**: MQTT message broker for handling publish/subscribe communications.
2. **React Client**: Frontend application serving as the MQTT client to send and receive messages.
3. **Docker**: Orchestrates the setup for both services.

What is MQTT?

MQTT is a lightweight messaging protocol designed for low-bandwidth, high-latency, or unreliable networks. It uses a publish/subscribe model, allowing clients to communicate via topics rather than direct addressing. It is widely used in IoT and real-time communication.

Key Features:

- **Publish/Subscribe Model**: Clients publish messages to a topic, and other clients subscribe to topics to receive messages.
- **Lightweight**: Minimal overhead makes it suitable for constrained devices and high-performance applications.
- **Quality of Service (QoS)**: Supports three levels to guarantee message delivery:
  - **QoS 0**: At most once.
  - **QoS 1**: At least once.
  - **QoS 2**: Exactly once.
- **Retained Messages**: Keeps the last message of a topic for new subscribers (toggleable).
- **Last Will and Testament (LWT)**: Sends a predefined message if a client disconnects unexpectedly, the message will be sent by the broker to the subscribed clients on behalf of the disconnected client.

Architecture

Mosquitto Broker

- Hosted on **Azure VM**.
- Configured with Docker Compose for ease of deployment and management.
- Exposes MQTT and WebSocket protocols for communication.

React App

- Hosted on a separate **Azure VM**.
- Connects to the Mosquitto broker as a client using an MQTT library (`mqtt.js`).

- Subscribes to topics and publishes messages via the broker.

Docker Compose Setup

1. Mosquitto Broker (Azure Server 1)

Use the `docker-compose.yml` file to deploy the Mosquitto broker:

```
services:
  mosquitto:
    image: eclipse-mosquitto:latest
    container_name: mosquitto
    ports:
      - "1883:1883"
      - "9001:9001"
    volumes:
      - ./mosquitto/config:/mosquitto/config
      - ./mosquitto/data:/mosquitto/data
      - ./mosquitto/log:/mosquitto/log
    restart: unless-stopped
```

Mosquitto Configuration File (`config/mosquitto.conf`):

```
listener 1883 # Enable MQTT support
listener 9001 # Enable WebSocket support
protocol websockets # Enable WebSocket protocol

allow_anonymous true # No authentication for simplicity

persistence true # Enable message persistence (not shure it's working on
the deployment)
persistence_file mosquitto.db
persistence_location /mosquitto/data/
```

Run the broker with:

```
docker-compose up -d
```

---

2. React Client (Azure Server 2, might be the same as the broker)

Use the `mqtt_demo/docker-compose.yml` file to host the React application:

```
services:
  mqtt-broker:
```

```
    image: your_docker_hub_url
    ports:
      - "80:80"
    volumes:
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
    restart: always
```

We now need to bundle our React app and serve it using Nginx. We can use a multi-stage Dockerfile to build the React app and serve it using Nginx.

This project uses BUN a zig-based build tool for building the React app.

Students from polytech made presentations about Zig. Pdf presentation here.

React Client Dockerfile:

```
# Stage 1: Build Stage
FROM oven/bun:latest AS base
WORKDIR /usr/src/app

# Copy app source code
COPY . .

# Install dependencies and build
RUN bun install
RUN bun add mqtt
RUN bun run build

# Stage 2: Run Stage (Nginx)

FROM nginx:alpine
COPY --from=base /usr/src/app/dist /usr/share/nginx/html

COPY ./nginx/default.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

Publish the image to Docker Hub

```
docker build -t your_docker_hub_url .
```

Example MQTT Integration in React:

Install the MQTT library:

```
npm install mqtt
```

Integrate MQTT in the React app:

```
const mqttClient = mqtt.connect(MQTT_BROKER);

mqttClient.on('connect', () => {
    mqttClient.subscribe(MQTT_TOPIC);
});

const handleMessage = (receivedTopic: string, payload: Buffer) => {
    // Handle incoming messages
};

client.on('message', handleMessage);
```

Deployment Steps

1. **Deploy Mosquitto Broker**:
    - Upload the `docker-compose.yml` and configuration files to Azure VM 1.
    - Run the broker using Docker Compose:

      ```
      docker-compose up -d
      ```

2. **Deploy React Client**:
    - Upload the React application code and `docker-compose.yml` to Azure VM 2.
    - Build and run the client using Docker Compose:

      ```
      docker-compose up -d
      ```

3. **Test Communication**:
    - Open the React app in a browser.
    - Interact with the MQTT broker by publishing/subscribing to topics.

Debugging

1. **Authentication**:
    - Configure Mosquitto to not require username/password or use TLS for secured connections. For the simplicity of this project, we have disabled authentication.

2. **Firewall Rules**:

    - Ensure Azure VMs allow traffic on MQTT (1883) and WebSocket (9001) ports. This one is important, as Azure VMs have a default firewall that blocks incoming traffic.

Future Enhancements

- Add support for **TLS/SSL encryption**.
- Use **Azure IoT Hub** as an alternative to Mosquitto for scalability.
- Implement more advanced client-side features like offline message storage.
- Integrate with **Azure Functions** for serverless processing of MQTT messages.
- Add **authentication** and **authorization** for secure communication.
- Implement **QoS** levels for message delivery guarantees.
- Use **Docker Swarm** or **Kubernetes** for container orchestration.
- Add **monitoring** and **logging** for better observability.