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 scalability.

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 Compose**: 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.
- **Last Will and Testament (LWT)**: Sends a predefined message if a client disconnects unexpectedly.

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)
Create a `docker-compose.yml` file to deploy the Mosquitto broker:

```yaml
version: '3.8'
services:
  mosquitto:
    image: eclipse-mosquitto:latest
    container_name: mosquitto_broker
    ports:
      - "1883:1883" # MQTT Port
      - "9001:9001" # WebSocket Port
    volumes:
      - ./config/mosquitto.conf:/mosquitto/config/mosquitto.conf
      - ./data:/mosquitto/data
      - ./log:/mosquitto/log
    restart: always
```

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

```
listener 1883
listener 9001
protocol websockets
allow_anonymous true
```

Run the broker with:

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

---

2. React Client (Azure Server 2)

Prepare a `docker-compose.yml` file to host the React application:

```
version: '3.8'
services:
  react-client:
    build:
      context: .
```
```

```
      dockerfile: Dockerfile
    container_name: react_client
    ports:
      - "3000:3000"
    restart: always
```

React Client Dockerfile:

```
# Use Node.js base image
FROM node:18

# Set working directory
WORKDIR /app

# Copy package.json and install dependencies
COPY package*.json ./
RUN npm install

# Copy application code
COPY . .

# Build React app
RUN npm run build

# Start React development server
CMD ["npm", "start"]
```

Example MQTT Integration in React:

Install the MQTT library:

```
npm install mqtt
```

Integrate MQTT in the React app:

```
import mqtt from 'mqtt';

const MQTT_URL = 'ws://<broker-public-ip>:9001'; // WebSocket connection

const client = mqtt.connect(MQTT_URL);

client.on('connect', () => {
  console.log('Connected to MQTT broker');

  // Subscribe to a topic
  client.subscribe('test/topic', (err) => {
```

```
    if (!err) {
      console.log('Subscribed to test/topic');
    }
  });

  // Publish a message
  client.publish('test/topic', 'Hello, MQTT!');
});

client.on('message', (topic, message) => {
  console.log(`Received message: ${message.toString()} on topic:
${topic}`);
});
```

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.

Security Considerations

1. **Authentication**:
   - Configure Mosquitto to require username/password or use TLS for secured connections.

2. **Firewall Rules**:
   - Ensure Azure VMs allow traffic on MQTT (1883) and WebSocket (9001) ports.

3. **Rate Limiting**:

    - Prevent misuse by limiting message throughput.

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.