HiveMQ Tutorial

Creating a Java application with Spring Boot that utilizes MQTT for messaging involves several steps, from setting up the project to integrating HiveMQ as the MQTT broker. This tutorial will guide you through the entire process, ensuring you have a fully functional application capable of sending and receiving messages using MQTT.

Prerequisites:

- JDK 11 or newer

- Maven or Gradle (this tutorial will use Maven)

- An IDE of your choice (Eclipse, IntelliJ IDEA, etc.)

- Docker (for running HiveMQ MQTT broker)

Step 1: Set up the Spring Boot project

Use Spring Initializr (https://start.spring.io/) to generate a new project. Choose Maven as the build tool, Java as the language, and the latest Spring Boot version. Add a dependency for Spring Web. Once configured, download the zip file and extract it to your workspace. Open the project in your IDE.

Step 2: Add the MQTT dependency

Edit the `pom.xml` file to add dependencies for the MQTT integration. Add the following dependency for the MQTT client, which provides a flexible and reliable way to connect to MQTT brokers:

```
<dependency>
   <groupId>com.hivemq</groupId>
   <artifactId>hivemq-mqtt-client</artifactId>
   <version>1.2.2</version>
</dependency>
```

Step 3: Run the MQTT Broker

To run HiveMQ as your MQTT broker, use Docker for simplicity:

docker run --name hivemq -d -p 1883:1883 hivemq/hivemq-ce

This command pulls the HiveMQ Community Edition image and runs it, exposing the default MQTT port 1883.

Step 4: Create an MQTT Configuration Class

Create a new Java class in your project for the MQTT configuration. This class will configure the MQTT client for connecting to HiveMQ.

```java
import com.hivemq.client.mqtt.lifecycle.MqttClientConnectedListener;
import com.hivemq.client.mqtt.mqtt5.Mqtt5AsyncClient;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.UUID;

@Configuration
public class MqttConfig {

    @Bean
    public Mqtt5AsyncClient mqttClient() {
        Mqtt5AsyncClient client = MqttClient.builder() MqttClientBuilder
                .useMqttVersion5() Mqtt5ClientBuilder
                .identifier(UUID.randomUUID().toString())
                .serverHost(System.getenv( name: "HIVEMQ_URI")) // "broker.hivemq.com"
                .serverPort(1883)
                .addConnectedListener(new MqttClientConnectedListener() {
                    @Override
                    public void onConnected(MqttClientConnectedContext context) {
                        System.out.println("Connected to the HiveMQ MQTT Broker!");
                    }
                })
                .automaticReconnect() MqttClientAutoReconnectBuilder.Nested<...>
                .initialDelay( l: 500, java.util.concurrent.TimeUnit.MILLISECONDS)
                .maxDelay( l: 1, java.util.concurrent.TimeUnit.MINUTES)
                .applyAutomaticReconnect() capture of ? extends Mqtt5ClientBuilder
                .buildAsync();
        return client;
    }
}
```

Step 5: Implement the MQTT Publisher

Create a service class to handle publishing messages to an MQTT topic:

```java
package com.ubbcluj.software_system_server.service;

import com.hivemq.client.mqtt.mqtt5.Mqtt5AsyncClient;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class MqttPublisherService {

    private final Mqtt5AsyncClient mqttClient;

    @Autowired
    public MqttPublisherService(Mqtt5AsyncClient mqttClient) { this.mqttClient = mqttClient; }

    public void publishMessage(String message) {
        // Check if the client is already connected
        if (!mqttClient.getState().isConnected()) {
            // Connect if not already connected
            mqttClient.connectWith()
                    .cleanStart(true)
                    .sessionExpiryInterval(500)
                    .send()
                    .whenComplete((connAck, throwable) -> {
                        if (throwable != null) {
                            System.out.println("Connection failed: " + throwable.getMessage());
                        } else {
                            publish(message); // Call publish method after successful connection
                        }
                    });
        } else {
            // Directly publish if already connected
            publish(message);
        }
    }

    private void publish(String message) {
        mqttClient.publishWith() Mqtt5PublishBuilder.Send<...>
                .topic( s: "SYSTEMCOMMUNICATION") Mqtt5PublishBuilder.Send<...>.Complete<...>
                .payload(message.getBytes())
                .send() CompletableFuture<Mqtt5PublishResult>
                .whenComplete((publishAck, pubThrowable) -> {
                    if (pubThrowable != null) {
                        System.out.println("Publish failed: " + pubThrowable.getMessage());
                    } else {
                        System.out.println("Message published to topic SYSTEMCOMMUNICATION");
                    }
                });
    }
}
```

Step 6: Implement the MQTT Subscriber

Create a class to subscribe to MQTT topics and handle incoming messages:

```java
package com.ubbcluj.software_system_backend.service;

import com.hivemq.client.mqtt.MqttClient;
import com.hivemq.client.mqtt.MqttGlobalPublishFilter;
import com.hivemq.client.mqtt.datatypes.MqttQos;
import com.hivemq.client.mqtt.mqtt5.Mqtt5AsyncClient;
import com.ubbcluj.software_system_backend.jpa.entity.Message;
import jakarta.annotation.PostConstruct;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;

@Service
public class MqttMessageConsumerService {

    @Autowired
    private MessageService messageService;

    private final JSONParser parser = new JSONParser();

    @PostConstruct
    public void subscribeToTopic() {
        Mqtt5AsyncClient client = MqttClient.builder() MqttClientBuilder
                .useMqttVersion5() Mqtt5ClientBuilder
                .identifier("consumer-id")
                .serverHost(System.getenv( name: "HIVEMQ_URI")) // "broker.hivemq.com"
                .serverPort(1883) // Default MQTT port
                .automaticReconnectWithDefaultConfig() // Enable automatic reconnect
                .buildAsync();

        client.connectWith()
                .cleanStart(true)
                .sessionExpiryInterval(500)
                .send()
                .whenComplete((connAck, throwable) -> {
                    if (throwable != null) {
                        System.out.println("Connection failed: " + throwable.getMessage());
                    } else {
                        System.out.println("Connected successfully");
                        // Subscribe to a topic
                        client.subscribeWith() Mqtt5AsyncClient.Mqtt5SubscribeAndCallbackBuilder.Start
                                .topicFilter( s: "SYSTEMCOMMUNICATION") // Change to your topic
                                .qos(MqttQos.AT_LEAST_ONCE) Mqtt5AsyncClient.Mqtt5SubscribeAndCallbackBuilder.Start.Complete
                                .send() CompletableFuture<Mqtt5SubAck>
                                .whenComplete((subAck, subThrowable) -> {
                                    if (subThrowable != null) {
                                        System.out.println("Subscribe failed: " + subThrowable.getMessage());
                                    } else {
                                        System.out.println("Subscribed successfully");
                                    }
                                });

                        // Set up callback for incoming messages
                        client.toAsync().publishes(MqttGlobalPublishFilter.ALL, publish -> {
                            try {
                                String message = new String(publish.getPayloadAsBytes());
                                JSONObject json = (JSONObject) parser.parse(message);
```

```
                        Message messageObj = new Message();
                        messageObj.setContent(json.get("message").toString());
                        messageObj.setCreatedAt(LocalDateTime.now());
                        messageService.saveMessage(messageObj);
                    } catch (Exception e) {
                        System.out.println("Failed to process message: " + e.getMessage());
                    }
                });
            }
        });
    }
}

```

Step 7: Create a Controller to test Publishing

Create a REST controller to test publishing messages via MQTT:

```
import java.util.List;

@RestController
@RequestMapping("/api")
public class MessageController {

    @Autowired
    public MessageService messageService;

    @Autowired
    private MqttPublisherService mqttPublisherService;

    @GetMapping("/data")
    public ResponseEntity<List<Message>> sendAllData() {
        System.out.println("Successfully received!");
        List<Message> messageList = messageService.findAllMessages();
        return ResponseEntity.ok(messageList);
    }

    @GetMapping("/send")
    public ResponseEntity<MessageResponse> sendMqtt() {
        System.out.println("Successfully received!");
        mqttPublisherService.publishMessage("{\"message\": \"Hello from the server.\"}");
        String message = "Successfully received!";
        MessageResponse response = new MessageResponse(message);
        return ResponseEntity.ok(response);
    }
}

```

Step 8: Running your application

- Run your Spring Boot application.

- Use a REST client or your browser to access `http://localhost:8080/send` to publish a message.

- Your `MqttSubscriber` should receive and print the message to the console.

Step 9: Testing

- Ensure your MQTT broker (HiveMQ) is running and accessible.

- Test publishing messages to various topics and ensure your subscriber receives them.

Conclusion

You have now created a simple Spring Boot application that can publish and subscribe to MQTT topics using HiveMQ as the broker. This setup demonstrates the basics of integrating MQTT into Java applications, which can be expanded for various IoT and messaging applications.