RabbitMQ Tutorial with Docker

This tutorial demonstrates how to set up **RabbitMQ** using **Docker**, and includes sample code for producing and consuming messages with RabbitMQ in a Java-based project.

Prerequisites:

- **Docker** installed on your machine.
- **Java Development Kit (JDK)** installed on your machine.
- **Maven** for managing dependencies and building the project.

Step 1: Setting up RabbitMQ with Docker

To run RabbitMQ, we will use **Docker** to quickly set up an instance of RabbitMQ on your machine.

Running RabbitMQ using Docker:

1. **Pull RabbitMQ Docker Image:** In your terminal, execute the following command to pull the official RabbitMQ image:

```
docker pull rabbitmq:management
```

This will pull the RabbitMQ image with the management plugin, which provides a web-based UI for monitoring and interacting with RabbitMQ.

2. **Run RabbitMQ Docker Container:** To run RabbitMQ with the default credentials, use the following command:

```
docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:management
```

- -p 5672:5672: Exposes RabbitMQ's default AMQP port.
- -p 15672:15672: Exposes the RabbitMQ management console.

3. **Verify RabbitMQ is Running:** After running the above command, you can check if RabbitMQ is running by executing:

```
docker ps
```

You should see the RabbitMQ container listed.

Step 2: RabbitMQ Docker Configuration

The RabbitMQ instance exposed on port 5672 allows AMQP clients to communicate with it, while the web management interface is available at port 15672. The default login credentials for RabbitMQ are:

- **Username:** guest
- **Password:** guest

Accessing RabbitMQ Management Interface:

To access the RabbitMQ Management UI, open your browser and go to:

```
http://localhost:15672
```

Log in with the default credentials (guest/guest).

Step 3: Project Setup

Create a new Java project and add the necessary dependencies in the pom.xml file (using Maven).

pom.xml:

```
<dependencies>
  <!-- RabbitMQ Client Dependency -->
  <dependency>
    <groupId>com.rabbitmq</groupId>
    <artifactId>amqp-client</artifactId>
    <version>5.13.0</version>
  </dependency>

  <!-- SLF4J Logging Dependency -->
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.30</version>
  </dependency>

  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>1.7.30</version>
  </dependency>
</dependencies>
```

application.properties:

```
spring.application.name=RabbitMQ_Tutorial
server.port = 8090
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

Step 4: Creating the Producer

The **Producer** will send messages to RabbitMQ.

MessageProducer.java:

```
package com.example.MessageProducer;

import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class MessageProducer {

   @Autowired
   private RabbitTemplate rabbitTemplate;

   public String publishMessage(String consumptionJSON) {
      rabbitTemplate.convertAndSend(MQConfig.EXCHANGE, MQConfig.ROUTING_KEY,
consumptionJSON);
      return "Message published!";
   }
}
```

Step 5: Creating the Consumer

The **Consumer** will listen for messages from RabbitMQ.

SimpleMessageConsumer.java:

```
package com.example.MessageProducer;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.ObjectMapper;
```

```
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;
@Component
public class SimpleMessageConsumer {

    private final ObjectMapper objectMapper = new ObjectMapper();

    @RabbitListener(queues = "energy-utility")
    public void consumeMessage(String consumptionJSON) {
        try {
            // Deserialize JSON to ConsumptionJSON object
            ConsumptionMessage consumption = objectMapper.readValue(consumptionJSON,
ConsumptionMessage.class);

            // Print the received message
            System.out.println("Received Consumption Data:");
            System.out.println("Date: " + consumption.getDate());
            System.out.println("Time: " + consumption.getTime());
            System.out.println("Energy Consumption: " + consumption.getEnergyConsumption());
            System.out.println("Metering Device ID: " + consumption.getMeteringDeviceId());
            System.out.println("-------------------------------------------------");
        } catch (JsonProcessingException e) {
            System.err.println("Failed to parse message: " + e.getMessage());
        }
    }
}
```

Step 6: Running the Producer and Consumer

1. **Start RabbitMQ Docker container** if it isn't already running:

```
docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:management
```

2. **Run the Consumer:** In the terminal, run the **SimpleMessageConsumer** class:

```
java com.example.SimpleMessageConsumer
```

3. **Run the Producer:** In another terminal, run the **MessageProducer** class:

```
java com.example.MessageProducer
```

You should see the **Producer** send a message and the **Consumer** receive it.

Step 7: Accessing RabbitMQ Management Interface

You can view the state of your queues, exchanges, and other RabbitMQ resources using the management interface.

1. Open your browser and go to http://localhost:15672.
2. Log in with the default credentials:
    - **Username**: guest
    - **Password**: guest
3. You should see the energy-utility queue under the "Queues" tab.