**Message brokers tutorial – Node.js& RabbitMQ**

The code in this tutorial can be found here: [github repository](https://github.com/anapapara/RabbitMQ-in-NodeJS-tutorial.git).

1. **Environment setup**

If you do not have installed on your computer **Node.js** and**Docker**, start by installing them.

* **Node.js**: [Download from here](https://nodejs.org/)
* **Docker:** [Download from here](https://docs.docker.com/engine/install/)

Then, check installation using the following commands:

A black screen with white text

AI-generated content may be incorrect.

1. **Create project structure**

|  |
| --- |
| message-brokers  ├── consumer  ├── producer  └──docker-compose.yml |

1. **RabbitMQ integration**
2. **Docker RabbitMQcontainer**

Add the following code inside the docker-compose.yml file:

|  |
| --- |
| version: '3'  services:    rabbitmq:      image: "rabbitmq:3-management"      container\_name: "rabbitmq\_container`"      ports:        - "5672:5672"        - "15672:15672"      environment:        RABBITMQ\_DEFAULT\_USER: guest        RABBITMQ\_DEFAULT\_PASS: guest |

Thedocker-compose.ymlfile is used to define and manage multi-container Docker applications. In our case, we use it to set up RabbitMQ easily. The file defines and runs RabbitMQ as a containerized service. It maps port 5672 (for messaging) and 15672 (for the web UI) while setting default credentials.

1. **Producer**

Firstly, we must initialize the nodeJS application and install necessary dependencies. So, run the following commands inside theproducerfolder:

|  |
| --- |
| cd producer  npm init -y  npm install amqplib dotenv |

Then, inside the producer folder, we create a.envfile with the following content:

|  |
| --- |
| RABBITMQ\_URL=amqp://guest:guest@localhost  QUEUE\_NAME=helloQueue |

You can rename your queue as you want. Here, our queue is calledhelloQueue.

The following step is to create a new file named producer.js, which contains the logic of the producer. Here the messages are ‘produced’ and sent to the RabbitMQqueue.

|  |
| --- |
| require('dotenv').config();  constamqp=require('amqplib');  asyncfunctionsendMessage() {    try {      // Connect to RabbitMQ      constconnection=awaitamqp.connect(process.env.RABBITMQ\_URL);      constchannel=awaitconnection.createChannel();      // Define queue name      constqueue=process.env.QUEUE\_NAME;      awaitchannel.assertQueue(queue, { durable:false });      // Message to send      constmessage="Hello from the Producer!";      channel.sendToQueue(queue, Buffer.from(message));      console.log(`Sent: ${message} on queue.`);      setTimeout(() => {        connection.close();        process.exit(0);      }, 500);    } catch (error) {      console.error("Error:", error);    }  }  sendMessage(); |

In this code, sendMessage() function is called to connect the producer to RabbitMQ queue and send messages. The connection is established through amqplib.connect() call and using the URL declared in .env file. Then, using channel.assertQueue() method, we create a queue with the name declared in .env. Setting the parameter durable to false, we assume that the messages do not persist so the queue will be gone when RabbitMQ shuts down. To send messages we utilize channel.sendToQueue(), having as parameters the name of the queue we want to send the message on and the message to send.

1. **Consumer**

Again, like in producer we initialize the nodeJS application and install the dependencies, but now in consumer folder.

|  |
| --- |
| cd consumer  npm init -y  npm install amqplib dotenv |

Then, create and edit the.envfile:

|  |
| --- |
| RABBITMQ\_URL=amqp://guest:guest@localhost  QUEUE\_NAME=helloQueue |

Here, the name of the queue must be the same as the one in producer so the two applications will use the same queue for communication.

Create file consumer.js and update it like below:

|  |
| --- |
| require('dotenv').config();  constamqp=require('amqplib');  asyncfunctionreceiveMessage() {      try {          // Connect to RabbitMQ          constconnection=awaitamqp.connect(process.env.RABBITMQ\_URL);          constchannel=awaitconnection.createChannel();          // Define queue name          constqueue=process.env.QUEUE\_NAME;          awaitchannel.assertQueue(queue, { durable:false });          console.log(`Waiting for messages in ${queue}...`);          // Receive messages          channel.consume(queue, (msg) => {              console.log(`Received: ${msg.content.toString()}`);              channel.ack(msg);          }, { noAck:false });      } catch (error) {          console.error("Error:", error);      }  }  receiveMessage(); |

Here, the consmer connects to RabbitMQ, creates a queue (same as producer) and listen for messages on that queuue using channel.consume(). When it recieves a message, it is processed (printed on console)and then acknowledged. By default, RabbitMQ automatically acknowledges messages ({ noAck: true }), meaning it removes them from the queue immediately after sending them to a consumer. However, we want to manually acknowledge messages to ensure they are properly processed before being removed.

Now, the project structure should look like:

|  |
| --- |
| message-brokers  │  ├── consumer  │ ├── .env  │ └── consumer.js  │  ├── producer  │ ├── .env  │ └── producer.js  |  └── docker-compose.yml |

These are the files created by us. The project will also contain files generated by npm: node\_modules folder and package-lock.json and package.json files.

1. **Run all**

Now, what is left is to run the producer and consumer and see their communication.

First, in the root directory run the following command to start RabbitMQ:

|  |
| --- |
| cd message-brokers  docker-compose up -d |

To check the RabbitMQ container is up and running you have to open Docker Desktop and shoud see the name of the container under the name of the project and the green bullet marking that it is running, like in the screenshot.

A screenshot of a computer

AI-generated content may be incorrect.

Another way to check is to access RabbitMQ Web UI in browser: <http://localhost:15672> and login using credentials guest / guest. You should see the RabbitMQ management dashboard and in ‘Queues and Streams’ page the queue helloQueue should appear with running status.

A screenshot of a computer

AI-generated content may be incorrect.

Then start the consumer using the commands:

|  |
| --- |
| cd consumer  node consumer.js |

You should see the message in console:

|  |
| --- |
| Waiting for messages in helloQueue... |

Open a new terminal for starting the producer:

|  |
| --- |
| cd producer  node producer.js |

Once the producer is started, the message should be printed on console and then the producer finishes.

|  |
| --- |
| Sent: Hello from the Producer! on queue. |
|  |

Returning to consumer console, there is printed the message marking that the message sent from the producer has been successfully received by the consumer.

|  |
| --- |
| Received: Hello from the Producer! |