

Rozproszone Systemy Informatyczne

Raport - Ćwiczenie 6

Paweł Kluska, 260391

Katsiaryna Ziatsikava, 245891

Kroki instalacji

1. Sprawdziliśmy jakie wersje RabbitMQ i Erlang współpracują ze sobą
2. Do zainstalowania wybraliśmy RabbitMQ - 3.11.17, Erlang - 25.0 .
3. Najpierw zainstalowaliśmy Erlang, instalator trzeba uruchomić z uprawnieniami Administratora.
4. Dalej zainstalowaliśmy RabbitMQ, instalator również uruchomiliśmy z uprawnieniami Administratora.
5. Uruchomiliśmy RabbitMQ Command Prompt (sbin dir)
6. W linii poleceń wpisaliśmy ***rabbitmq-plugins.bat enable rabbitmq_management***. To polecenie uruchomiło plik z pluginami.
7. Aby uruchomić broker RabbitMQ użyliśmy polecenie ***rabbitmq-server.bat***
8. Po uruchomieniu brokera RabbitMQ otworzyliśmy przeglądarkę i przeszliśmy do RabbitMQ Management Console, dostępnego pod adresem:
<http://localhost:15672> .
9. Zalogowaliśmy się do RabbitMQ Management Console. Domyślny login i hasło to "***guest/guest***".
10. Aby uruchomić w konfiguracji dwumaszynowej, w zakładce Admin dodaliśmy nowego użytkownika, ustawiliśmy go jako administrator i nadaliśmy uprawnienia.

Aplikacja

Aby uruchomić aplikację, dodaliśmy bibliotekę RabbitMQ Java Client, która dostarcza niezbędne klasy do obsługi komunikacji z RabbitMQ. Bibliotekę dodaliśmy przy pomocy narzędzia do budowania projektów Maven.

Kod **nadawcy** wygląda następująco:

```
public class Sender {  
  
    4 usages  
    private final static String QUEUE_NAME = "hello";  
  
    public static void main(String[] argv) throws Exception {  
        MyData.info();  
        Runnable sender1 = getSender1();  
        Runnable sender2 = getSender2();  
        Thread thread1 = new Thread(sender1);  
        Thread thread2 = new Thread(sender2);  
  
        thread1.start();  
        thread2.start();  
    }  
}
```

```

1 usage
public static Runnable getSender1() {
    Runnable basic = () ->
    {
        Random r = new Random();
        long min = 1500;
        long max = 2500;
        long random = (long) (min + r.nextDouble() * (max - min));
        ConnectionFactory factory = new ConnectionFactory();
        factory.setUsername("pawel");
        factory.setPassword("pawel");
        factory.setHost("10.108.127.202");

        try (Connection connection = factory.newConnection();
            Channel channel = connection.createChannel()) {
            channel.queueDeclare(QueueName, b: false, b1: false, b2: false, map: null);

            for (int i = 0; i < 5; i++) {
                String message = "Pawel " + i;
                String fullMessage = Thread.currentThread().getName() + ": " + message + " ";
                channel.basicPublish("", QueueName, basicProperties: null, fullMessage.getBytes(StandardCharsets.UTF_8));
                Thread.sleep(random);
            }
        } catch (IOException | TimeoutException | InterruptedException e) {
            throw new RuntimeException(e);
        }
    };
    return basic;
}

```

```

1 usage
public static Runnable getSender2() {
    Runnable basic = () ->
    {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setUsername("pawel");
        factory.setPassword("pawel");
        factory.setHost("10.108.127.202");

        Random r = new Random();
        long min = 2000;
        long max = 3500;
        long random = (long) (min + r.nextDouble() * (max - min));

        try (Connection connection = factory.newConnection();
            Channel channel = connection.createChannel()) {
            channel.queueDeclare(QueueName, b: false, b1: false, b2: false, map: null);

            for (int i = 0; i < 5; i++) {
                String message = "Katya " + i;
                String fullMessage = Thread.currentThread().getName() + ": " + message + " ";
                channel.basicPublish("", QueueName, basicProperties: null, fullMessage.getBytes(StandardCharsets.UTF_8));
                Thread.sleep(random);
            }
        } catch (IOException | TimeoutException | InterruptedException e) {
            throw new RuntimeException(e);
        }
    };
    return basic;
}

```

Sender miał działać wg następujących założeń. Definiujemy 2 senderów, którzy wysyłają jednocześnie wiadomości do brokera RabbitMq. Każdy z nich ma swoje połączenie do brokera oraz inny przedział czasowy, z którego losujemy liczbę, którą

muszą odczekać senderzy pomiędzy wysłaniem każdej z wiadomości. Te przedziały różnią się pomiędzy senderami, jednak mogą one zachodzić na siebie. Dzięki temu może się zdarzyć, że jeden z senderów wyśle 2 wiadomości pod rząd. W celu osiągnięcia powyższych założeń zdecydowaliśmy się użyć mechanizmu wielowątkowości.

Senderzy podłączają się do brokera, który jest uruchomiony na drugiej maszynie. Przy konfiguracji połączenia zostały podane: adres ip, nazwa użytkownika, hasło, oraz nazwa kolejki, do której sender chce wysłać wiadomości.

Kod nadawcy składa się z 3 części.

- Metody tworzącej sendera 1 - pierwszy wątek
- Metody tworzącej sendera 2 - drugi wątek
- Metody main, która tworzy wcześniej zdefiniowane wątki i je uruchamia.

Kod odbiorcy wygląda następująco:

```
public class Client {  
  
    2 usages  
    private final static String QUEUE_NAME = "hello";  
  
    public static void main(String[] argv) throws Exception {  
        MyData.info();  
        ConnectionFactory factory = new ConnectionFactory();  
        factory.setUsername("pawel");  
        factory.setPassword("pawel");  
        factory.setHost("10.108.127.202");  
        Connection connection = factory.newConnection();  
        Channel channel = connection.createChannel();  
  
        channel.queueDeclare(QUEUE_NAME, b: false, b1: false, b2: false, map: null);  
        System.out.println("Oczekiwanie na wiadomości...");  
  
        DeliverCallback deliverCallback = (consumerTag, delivery) -> {  
            String message = new String(delivery.getBody(), StandardCharsets.UTF_8);  
            System.out.println("Otrzymano wiadomość: " + message + "");  
        };  
        channel.basicConsume(QUEUE_NAME, b: true, deliverCallback, consumerTag -> { });  
    }  
}
```

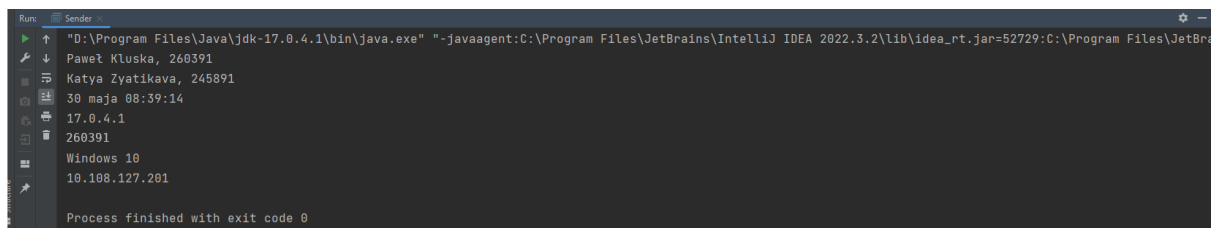
W kodzie wyżej stworzyliśmy stałą QUEUE_NAME o wartości "hello", która jest nazwą kolejki, z której będą odbierane wiadomości. Skonfigurowaliśmy połączenia z RabbitMQ, ustawiliśmy adres hosta, nawiązaliśmy połączenie z RabbitMQ. Stworzyliśmy kanał komunikacyjny wewnątrz połączenia. Za pomocą

DeliverCallback wiadomość jest odbierana i wyświetlana w konsoli. Na sam koniec uruchamiamy konsumenta, który zaczyna odbierać wiadomości z kolejki “hello”

Działanie aplikacji

Aplikacja została uruchomiona w konfiguracji dwumaszynowej: senderzy są uruchomieni na jednej maszynie, klient oraz serwer RabbitMQ na drugiej.

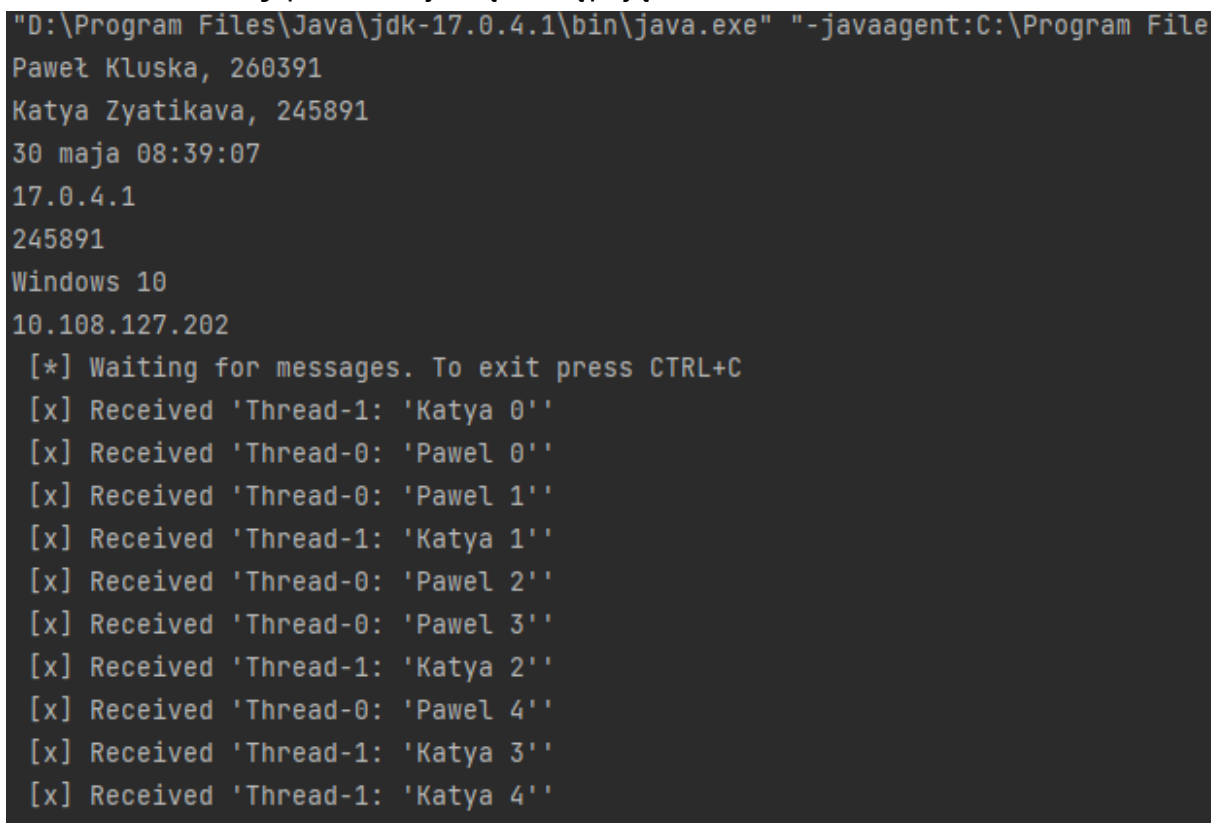
Działanie **nadawcy** prezentuje się następująco:



```
Run: Sender
"D:\Program Files\Java\jdk-17.0.4.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2022.3.2\lib\idea_rt.jar=52729:C:\Program Files\JetBrains\IntelliJ IDEA 2022.3.2\bin" -Dfile.encoding=UTF-8
Paweł Kluska, 260391
Katya Zyatikava, 245891
30 maja 08:39:14
17.0.4.1
260391
Windows 10
10.108.127.201
Process finished with exit code 0
```

Tutaj tylko została uruchomiona metoda info z klasy MyData.

Działanie **odbiorcy** prezentuje się następująco:



```
"D:\Program Files\Java\jdk-17.0.4.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2022.3.2\lib\idea_rt.jar=52729:C:\Program Files\JetBrains\IntelliJ IDEA 2022.3.2\bin" -Dfile.encoding=UTF-8
Paweł Kluska, 260391
Katya Zyatikava, 245891
30 maja 08:39:07
17.0.4.1
245891
Windows 10
10.108.127.202
[*] Waiting for messages. To exit press CTRL+C
[x] Received 'Thread-1: 'Katya 0''
[x] Received 'Thread-0: 'Pawel 0''
[x] Received 'Thread-0: 'Pawel 1''
[x] Received 'Thread-1: 'Katya 1''
[x] Received 'Thread-0: 'Pawel 2''
[x] Received 'Thread-0: 'Pawel 3''
[x] Received 'Thread-1: 'Katya 2''
[x] Received 'Thread-0: 'Pawel 4''
[x] Received 'Thread-1: 'Katya 3''
[x] Received 'Thread-1: 'Katya 4''
```

Widać tutaj jak po kolei zostają wysłane wiadomości przez senderów. Nie są one wysyłane po kolei, tylko według wylosowanych przedziałów czasowych.

Protokół AMQP (Advanced Message Queuing Protocol)

RabbitMQ wykorzystuje protokół AMQP do komunikacji między klientami i serwerem. AMQP (Advanced Message Queuing Protocol) to standardowy protokół komunikacyjny do asynchronicznej wymiany wiadomości między systemami. Jest niezawodny, elastyczny i skalowalny. Pozwala na bezpieczne przesyłanie wiadomości w architekturach opartych na kolejce. Jest używany w systemach kolejek wiadomości, przetwarzania strumieniowego i integracji aplikacji. W naszym przypadku nadawca generuje wiadomości i wysyła je do kolejek, a odbiorca odbiera i przetwarza te wiadomości.