

# RabbitMQ

## Instrukcja do ćwiczeń laboratoryjnych

Filip Malawski | [fmal@agh.edu.pl](mailto:fmal@agh.edu.pl) | 03.2021

### 1. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie z zagadnieniami związanymi z wykorzystaniem komponentów typu Message Oriented Middleware, które umożliwiają komunikację przez wymianę wiadomości. Zagadnienia prezentowane są na przykładzie platformy RabbitMQ.

Wynikiem ćwiczenia ma być raport w formie **dokumentu PDF** zawierający: **imię, nazwisko, numer indeksu** oraz odpowiedzi do pytań (w opisie zadań podano co ma się znaleźć w raporcie). Raport należy przesłać przez UPEL (RabbitMQ - zadanie z ćwiczeń). **Nazwa dokumentu ma być zgodna ze schematem: Nazwisko\_Imie\_rabbitmq\_lab.pdf**, np. *Mickiewicz\_Adam\_rabbitmq\_lab.pdf*.

### 2. Wprowadzenie do ćwiczenia

Dwa podstawowe modele komunikacji w systemach rozproszonych to komunikacja synchroniczna oraz asynchroniczna.

Komunikacja synchroniczna:

- Obie uczestniczące strony muszą być aktywne
- Wywołania blokujące

Komunikacja asynchroniczna:

- Obie strony nie muszą być jednocześnie aktywne
- Wywołania nieblokujące
- Potwierdzenie odbioru (opcjonalnie)

Komunikacja przez wiadomości jest asynchroniczną alternatywą do synchronicznego wywoływania zdalnych metod. Cechy komunikacji przez wiadomości:

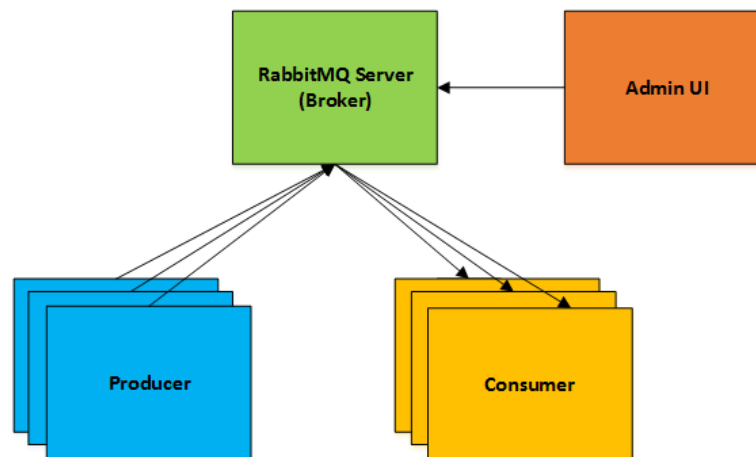
- Format wiadomości definiuje sposób przesyłania danych (tak jak interfejs w metodach)
- Ukierunkowanie na zdarzenia
- Brak sztywnych zależności czasowych
- Luźne powiązania komponentów

Message Oriented Middleware (MOM) jest to warstwa pośrednia dostarczająca mechanizmów obsługi komunikacji z użyciem wiadomości. Przykładem MOM jest platforma RabbitMQ.

Główne cechy RabbitMQ:

- Mechanizmy wyboru ścieżek do obsługi wiadomości (routing)
- Mechanizmy zapewnienia niezawodności (potwierdzenia, ponowne wysyłanie)
- Wsparcie dla różnych protokołów
- Wsparcie dla różnych języków programowania
- Interfejs do zarządzania
- Pluginy

Podstawowe elementy składowe platformy RabbitMQ, przedstawione zostały na poniższym schemacie:



- Server (Broker) – przyjmuje oraz przekazuje wiadomości, zapewnia routing oraz niezawodność
- Producer – produkuje wiadomości, czyli wysyła je do serwera
- Consumer – konsumuje wiadomości, czyli otrzymuje je od serwera
- Admin UI – interfejs do zarządzania serwerem przez przeglądarkę

### 3. Program ćwiczenia

Na platformie UPEL znajdują się materiały do ćwiczenia: kody źródłowe oraz biblioteki klienta RabbitMQ. Przed rozpoczęciem ćwiczenia zainstaluj Erlanga <https://www.erlang.org/downloads> oraz serwer RabbitMQ <https://www.rabbitmq.com/download.html> Ćwiczenia należy wykonać w języku **Java**.

1. Stwórz projekt w Javie, umieść w nim załączone kody źródłowe oraz dołącz biblioteki. Zapoznaj się z kodami źródłowymi Z1\_Consumer oraz Z1\_Producer, które realizują najprostszy przypadek komunikacji z użyciem pojedynczej kolejki na wiadomości
  - Zwróć uwagę na etap nawiązywania połączenia
  - Zwróć uwagę na deklarację kolejki u producenta oraz konsumenta
  - Zwróć uwagę na sposób wysyłania oraz odbierania wiadomości
2. Uruchom serwer RabbitMQ
  - (Windows) Menu start => RabbitMQ Service – start
3. Uruchom przykład Z1
  - Uruchom konsumenta Z1\_Consumer
  - Uruchom producenta Z1\_Producer
  - Powinna zostać przesłana wiadomość
4. Uruchom plugin administracyjny oraz zapoznaj się z konsolą administracyjną RabbitMQ
  - (Windows) Menu start => RabbitMQ Command Prompt
  - Wpisz: `rabbitmq-plugins enable rabbitmq_management`
  - Otwórz konsolę dostępną pod adresem: <http://localhost:15672/>  
user: guest, password: guest
  - Zaobserwuj śledzenie przesyłu wiadomości

5. RabbitMQ oferuje szereg mechanizmów obsługi kolejek:
- Potwierdzenia po otrzymaniu lub po przetworzeniu wiadomości
  - Dystrybucja wiadomości do wielu konsumentów – domyślnie round-robin, możliwe równoważenie obciążenia (load-balancing)
  - Trwałość – możliwość zachowania wiadomości przy restarcie serwera
6. **Zad 1a (1 punkt). Zaobserwuj działanie mechanizmu niezawodności w różnych scenariuszach potwierdzeń wiadomości.**
- Zmodyfikuj producenta, aby wysyłał wiadomości wpisane z konsoli:  

```
BufferedReader br = new BufferedReader(new  
InputStreamReader(System.in));
```
  - Zmodyfikuj konsumenta, aby obsługiwał wiadomość przez zadany czas (podany w sek.)  

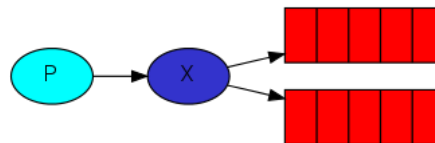
```
int timeToSleep = Integer.parseInt(message);  
Thread.sleep(timeToSleep * 1000);
```
  - Sprawdź działanie potwierdzeń:
    - Po otrzymaniu wiadomości:  

```
channel.basicConsume(QueueName, true, consumer);
```
    - Po przetworzeniu wiadomości  

```
channel.basicAck(envelope.getDeliveryTag(), false)  
(...)  
channel.basicConsume(QueueName, false, consumer);
```
- w następujących scenariuszach:
- Konsument zostaje zrestartowany po zakończeniu przetwarzania wiadomości
  - Konsument zostaje zrestartowany w trakcie przetwarzania wiadomości
- Odpowiedz na pytania:
    - Który sposób potwierdzeń zapewnia większą niezawodność?
    - Co się stanie, jeśli nie będziemy potwierdzać wiadomości ani po otrzymaniu, ani po przetworzeniu?
  - **W raporcie należy umieścić:** odpowiedzi na powyższe 2 pytania
7. **Zad 1b (1 punkt). Zaobserwuj działanie mechanizmu load-balancing**
- Ustaw potwierdzenia po przetworzeniu wiadomości (**ważne!**)
  - Uruchom dwóch konsumentów oraz jednego producenta
  - Wyślij od producenta 10 wiadomości: na przemian krótkie (1 s.) oraz długie (5 s.) zadania (1,5,1,5,1,5,1,5,1,5)
  - Zaobserwuj rozłożenie zadań pomiędzy konsumentów (domyślny algorytm to round-robin)
  - Dodaj obsługę QoS u konsumentów: `channel.basicQos(1)`
  - Ponownie wyślij od producenta 10 wiadomości: na przemian krótkie (1 s.) oraz długie (5 s.) zadania
  - Zaobserwuj zmianę w działaniu rozłożenia wiadomości
  - Zapoznaj się z opisem działania metody `basicQos` w dokumentacji RabbitMQ
  - **W raporcie należy umieścić:** output wypisany przez obu konsumentów przed oraz po włączeniu obsługi QoS

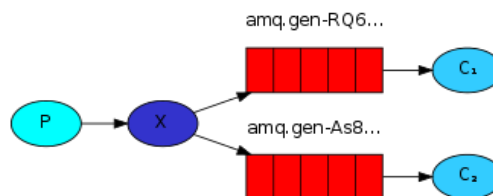
## 8. Pojęcie Exchange w RabbitMQ

- Producent nie wysyła wiadomości bezpośrednio do kolejki, lecz do Exchange
- Exchange decyduje, do których kolejek wysłać wiadomość (**może wysłać kopię wiadomości do więcej niż jednej kolejki**)
- Wiadomości z kolejek trafiają do konsumentów (**jedna wiadomość z kolejki trafia do jednego konsumenta, nawet jeśli do kolejki zapisanych jest ich więcej** – patrz Zad. 1b)
- Różne typy Exchange (Direct, Topic, Fanout) pozwalają na różne konfiguracje routingu wiadomości
- Dotychczas korzystaliśmy z domyślnego Exchange, które nie ma nazwy i nie jest podawane wprost
- Uwaga: przy korzystaniu z Exchange innego niż domyślne, należy związać (bind) kolejkę z danym Exchange, aby otrzymywać z niego wiadomości
- Na rysunku poniżej: P – producent, X - exchange



## 9. Routing Fanout

- Model Publish/Subscribe
- Każda kolejka związana do danego Exchange dostaje kopię wiadomości



- Wiązanie kolejki z Exchange:  

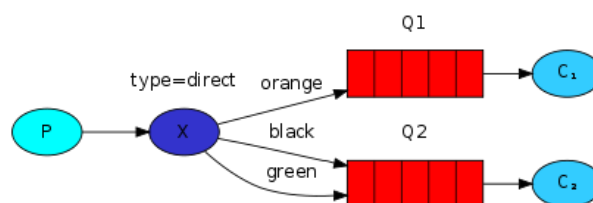
```
String queueName = channel.queueDeclare().getQueue();  
channel.queueBind(queueName, EXCHANGE_NAME, "");
```

## 10. Routing Direct

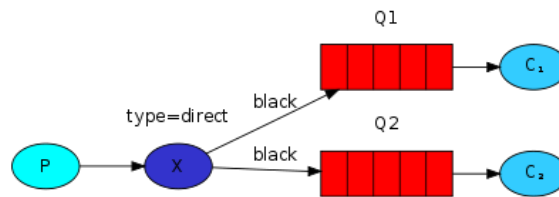
- Przy wiązaniu kolejki z Exchange podajemy klucz  

```
channel.queueBind(queueName, EXCHANGE_NAME, routingKey);
```
- Przy wysłaniu wiadomości również podajemy klucz  

```
channel.basicPublish(EXCHANGE_NAME, routingKey, ...)
```
- Tylko wiadomości ze zgodnym kluczem trafią do kolejki
- Możliwe wiele kluczy dla jednej kolejki:

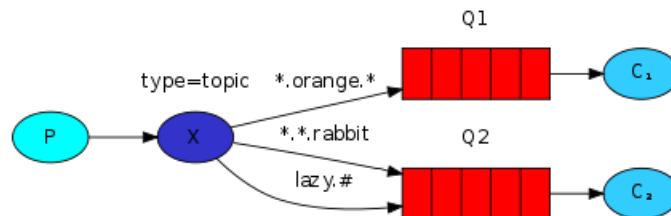


- Możliwe wiele kolejek z tym samym kluczem:

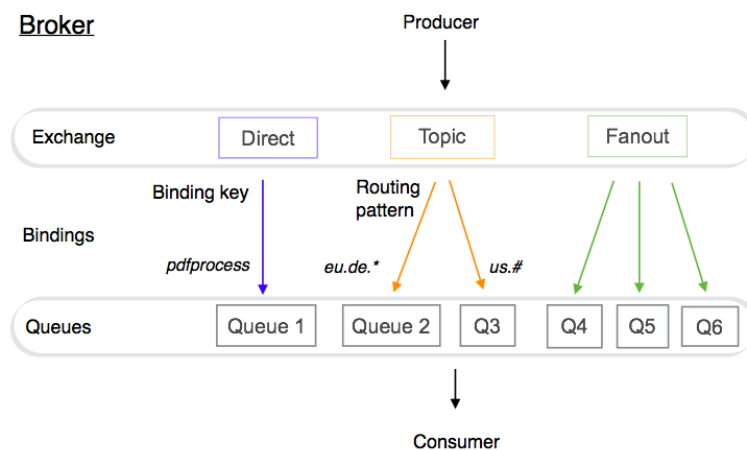


### 11. Routing Topic

- Klucz z dopasowaniem do wzorca, możliwe znaki specjalne
- np. blue.fast.\*.sedan.#
- \* to dokładnie jedno dowolne słowo
- # to zero lub więcej dowolnych słów
- Uwaga – znaki specjalne mogą być użyte **tylko** przy wiązaniu kolejki (nie mogą być użyte jako **klucz w wiadomości**)



### 12. Routing – podsumowanie



### 13. Zapoznaj się z kodami źródłowymi Z2\_Producer oraz Z2\_Consumer

- Zwróć uwagę na operacje `exchangeDeclare` oraz `queueBind`
- Zwróć uwagę, że każdy konsument tworzy swoją kolejkę (nazwa generowana automatycznie) i wiąże ją ze wspólnym Exchange typu Fanout

### 14. Uruchom przykład Z2 (routing Fanout)

- Uruchom producenta Z2\_Producer
- Uruchom dwóch konsumentów Z2\_Consumer
- Wyślij wiadomość od producenta
- Każdy konsument powinien dostać kopię wiadomości

**15. Zad 2 (2 punkty). Zaimplementuj oraz pokaż w działaniu routing Direct oraz Topic**

- Zmodyfikuj producenta tak, aby przyjmował z konsoli klucz routingu oraz wiadomość
- Zmodyfikuj konsumenta tak, aby przyjmował z konsoli klucz routingu
- Ustaw odpowiedni typ Exchange po obu stronach
- Raz zadeklarowany Exchange nie może zmienić swojego typu – aby skorzystać z innego typu Exchange zadeklaruj Exchange o innej nazwie lub usuń poprzedni Exchange (np. przez konsolę administracyjną)
- Zaproponuj przykłady, które zobrazują zasadę oraz różnice w działaniu routingu Direct oraz Topic
- **W raporcie należy umieścić:** opis zaproponowanych przykładów dla routingu Direct oraz Topic (klucze przy wiązaniu kolejek, klucze w przesłanych wiadomościach, kto dostanie przykładowe wiadomości)

**16. Wskazówki do zadania 2:**

```
channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.DIRECT);
channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.TOPIC);
channel.basicPublish(EXCHANGE_NAME, key, null, message.getBytes("UTF 8"));
channel.queueBind(queueName, EXCHANGE_NAME, key);
```

## 4. Literatura

- <https://www.rabbitmq.com/getstarted.html>

## Załączniki

- Załączniki do ćwiczeń znajdują się na platformie UPEL w kursie Systemy Rozproszone