Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
AGH
Unia Europejska Europejski Fundusz Społeczny

Zintegrowany Program Rozwoju
Akademii Górniczo-Hutniczej w Krakowie
Nr umowy: **POWR.03.05.00-00-Z307/17**

Instrukcja do ćwiczeń laboratoryjnych

Nazwa przedmiotu	Systemy rozproszone
Numer ćwiczenia	3
Temat ćwiczenia	Message Oriented Middleware - RabbitMQ

Poziom studiów	I stopień
Kierunek	Informatyka
Forma i tryb studiów	Stacjonarne
Semestr	6

dr inż. Filip Malawski

Wydział Informatyki, Elektroniki i Telekomunikacji

Kraków, 2020

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 mają być **kody źródłowe** do zadań (w języku Java) oraz **raport** w pliku tekstowym lub PDF zawierający: imię, nazwisko, numer indeksu oraz odpowiedzi do pytań (w opisie zadań podano co ma się znaleźć w raporcie). Szczegóły odnośnie przesyłania rozwiązań znajdują się w punkcie 4. instrukcji.

2.Wprowadzenie do ćwiczenia

Dwa podstawowe modele komunikacji w systemach rozproszonych do 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

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

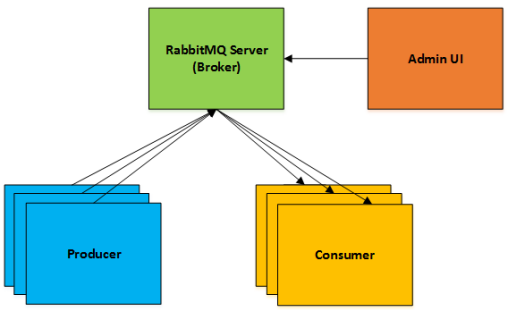
- Format wiadomości definiuje sposób przesłania danych (tak jak interfejs w metodach)
- Ukierunkowanie na zdarzenia
- Brak sztywnych zależności czasowych
- Luźne powiązania komponentów

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

Elementy składowe platformy RabbitMQ, przedstawione 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(QUEUE_NAME, true, consumer);
       ```
     - Po przetworzeniu wiadomości
       ```
       channel.basicAck(envelope.getDeliveryTag(), false)
       (…)
       channel.basicConsume(QUEUE_NAME, false, consumer);
       ```

     w następujących scenariuszach:
     - Konsument zostaje zrestartowany po zakończeniu przetwarzania wiadomości
     - Konsument zostaje zrestartowany w trakcie przetwarzania wiadomości

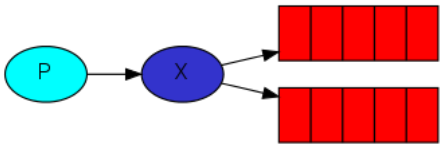
- Uzupełnij poniższą tabelę, wpisując dla każdego przypadku, czy konsument zakończy obsługę wiadomości (wpisz TAK lub NIE)

Potwierdzenie	Restart po przetworzeniu wiadomości	Restart w trakcie przetwarzania wiadomości
Po otrzymaniu wiadomości		
Po przetworzeniu wiadomości		

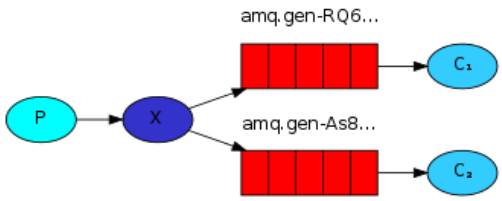
- Który sposób potwierdzeń zapewnia większą niezawodność?
- Co się stanie, jeśli nie będziemy potwierdzać wiadomości ani po otrzymaniu, ani po przetworzeniu?
- **W raporcie należy umieścić**: uzupełnioną tabelę oraz odpowiedzi na powyższe 2 pytania

7. **Zad 1b (1 punkt). Zaobserwuj działanie mechanizmu load-balancing'u**
   - Ustaw potwierdzenia po przetworzeniu wiadomości
   - 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, "");
```

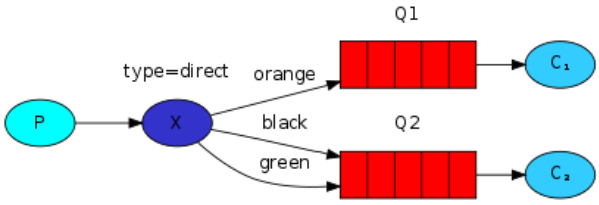
10. Routing Direct
   - Przy wiązaniu kolejki z Exchange podajemy klucz

```
channel.queueBind(queueName, EXCHANGE_NAME, routingKey);
```

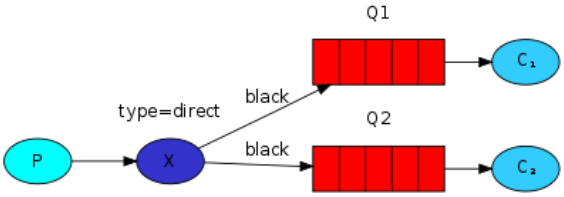
   - Przy wysyłaniu wiadomości również podajemy klucz

```
channel.basicPublish(EXCHANGE_NAME, routingKey, …)
```

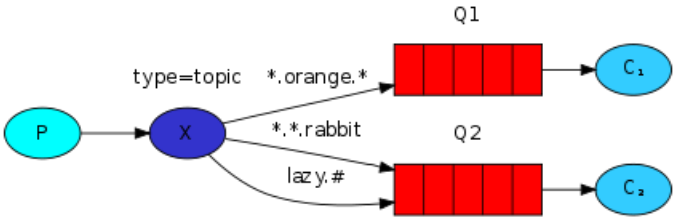
   - Tylko wiadomości ze zgodnym kluczem trafią do kolejki
   - Możliwe wiele kluczy dla jednej kolejki



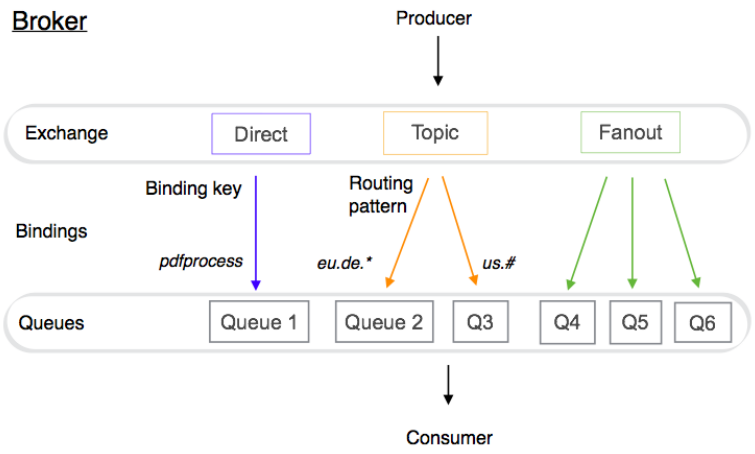
   - Możliwe wiele kolejek z tym samym kluczem



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 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. Sposób oceny

Ocena z ćwiczeń lab. 0 – 5 punktów:

- 1 punkt za obecność na zajęciach (20% oceny)
- 4 punkty za wykonane zadania (80% oceny)

Rozwiązanie ćwiczenia należy przesłać w formie pojedynczego archiwum ZIP o nazwie w formacie: *Nazwisko_Imie_rabbitmq_lab*, np. *Mickiewicz_Adam_rabbitmq_lab*.

W archiwum mają się znaleźć następujące pliki:

- kod źródłowy do zadania 1a
- kod źródłowy do zadania 1b
- kod źródłowy do zadania 2 (wystarczy kod dla routingu Topic)
- raport w pliku tekstowym lub PDF z imieniem, nazwiskiem, numerem indeksu oraz odpowiedziami do zadań:
    - 1a – tabela oraz odpowiedzi na 2 pytania
    - 1b – output od konsumentów w dwóch przypadkach
    - 2 – przykłady dla routingu Direct oraz Topic

Archiwum należy umieścić na platformie UPEL do czasu zakończenia zajęć swojej grupy.

5. Literatura

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

Załączniki

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