RabbitMQ – otwarty system pośredniczący w wymianie wiadomości w środowisku rozproszonym

RabbitMQ jest czymś w rodzaju firmy pocztowej, która jest organizacją złożoną nie tylko z placówek pocztowych, ale i całej infrastruktury służącej do przekazywania wiadomości z jednego miejsca na drugie, od nadawcy do odbiorcy. W RabbitMQ nadawca zwany jest producentem (ang. producer), odbiorca konsumentem (ang. consumer), a wspomniana firma pocztowa to w dużym uproszczeniu kolejka wiadomości (ang. message queue). Wiadomości są przesyłane w środowisku rozproszonym, co oznacza, że „nasza firma pocztowa" będzie odpowiedzialna za dostarczanie pakietów danych pomiędzy różnymi aplikacjami, często złożonymi systemami.

Wiadomością (pakietem danych) może być dowolny ciąg bajtów, a zatem mogą to być dane w formie tekstowej: w formacie JSON, pliki zakodowane w formie base64, proste wiadomości tekstowe czy pojedyncze linie tekstu, np. logi serwera czy konkretnej aplikacji. Mogą to również być dane binarne, np. plik z obrazkiem czy piosenką w formacie mp3.

Dzięki wspólnemu pośrednikowi wiadomości, nasze programy mogą efektywnie komunikować się ze sobą, specjalizując się w wykonywaniu określonych czynności na przesyłanych informacjach.

Na potrzeby tego artykułu serwer RabbitMQ [1] będzie nazywany pośrednikiem wiadomości oraz kurierem wiadomości.

Ponieważ opisywany system potrafi przetwarzać od kilkudziesięciu tysięcy wiadomości na sekundę (w zależności od mocy obliczeniowej serwera) czy nawet miliona na platformie Google Cloud Platform [2], istotne jest, by pakiety danych były nieduże, ponieważ wspomniana szybkość wiąże się z tym, że dane przechowywane są w pamięci RAM. Przesyłanie do kolejki grafik, każdej o rozmiarze 100 MB, bardzo szybko zablokowałoby możliwości RabbitMQ. Kurier wiadomości gromadzi pakiety danych i w zależności od rodzaju kolejki, a także jej konfiguracji – może czekać, aż pojawi się ich odbiorca lub odbiorcy; może przekazywać otagowane wiadomości tylko do odpowiednich konsumentów lub nawet odrzucać je, jeśli nikt się po nie nie zjawi.

WSTĘP DO ARTYKUŁU

Artykuł ma na celu zaprezentowanie zarówno wiedzy teoretycznej związanej z systemami pośredniczącymi w wymianie wiadomości w oparciu o protokół Advanced Message Queuing Protocol (w skrócie AMQP) na przykładzie serwera RabbitMQ, jak i wiedzy praktycznej pokazującej możliwości poszczególnych rodzajów obsługi kolejek opisanych w tym protokole i zwanych centralami wiadomości. Część praktyczna zostanie zaprezentowana na przykładach wykorzystujących serwer node.js, jednak większość popularnych języków programowania i ich frameworków posiada wsparcie dla protokołu AMQP, dzięki czemu w oparciu o serwer RabbitMQ można stworzyć platformę integrującą aplikacje napisane w językach Java, PHP i np. frameworku Ruby on Rails.

Język Erlang

Język Erlang został zaprojektowany pod kątem aplikacji współbieżnych uruchamianych w środowisku rozproszonym, gdzie szczególnie ważne są takie cechy jak skalowalność, odporność na awarie i długotrwała praca przy dużej liczbie wątków.

Powstał on jeszcze w latach 80-tych, a jego twórcą jest Joe Armstrong – pracownik firmy Ericsson. Nazwę tego języka można wziąć za skrót od dwóch angielskich słów: *Ericsson* oraz *language*, jednakże określenie Erlang zostało nadanae na cześć pierwszego naukowca w dziedzinie telekomunikacji: Agnera Krarupa Erlanga (1878-1929), matematyka duńskiego, pioniera teorii ruchu telekomunikacyjnego oraz teorii kolejek.

Erlang to język funkcyjny, dynamiczny, o ścisłym typowaniu, w którym nie istnieje pojęcie wątku – wszystkie zadania są procesami. Erlang obsługuje mechanizm tzw. hot-swapping'u, pozwalający na aktualizację kodu aplikacji bez jej zatrzymywania. Ma to szczególne znaczenie w urządzeniach telekomunikacyjnych, które muszą pracować 24h na dobę, 7 dni w tygodniu, 365/366 dni w roku. Erlang potrafi wykorzystywać możliwości maszyn wieloprocesorowych oraz wielordzeniowych, a także skalować aplikację na środowisko sieciowe, rozproszone w sposób przezroczysty – model współbieżności sprawia, że z punktu widzenia procesu nie ma różnicy między komunikacją z innym procesem uruchomionym na tym samym, czy na odległym węźle.

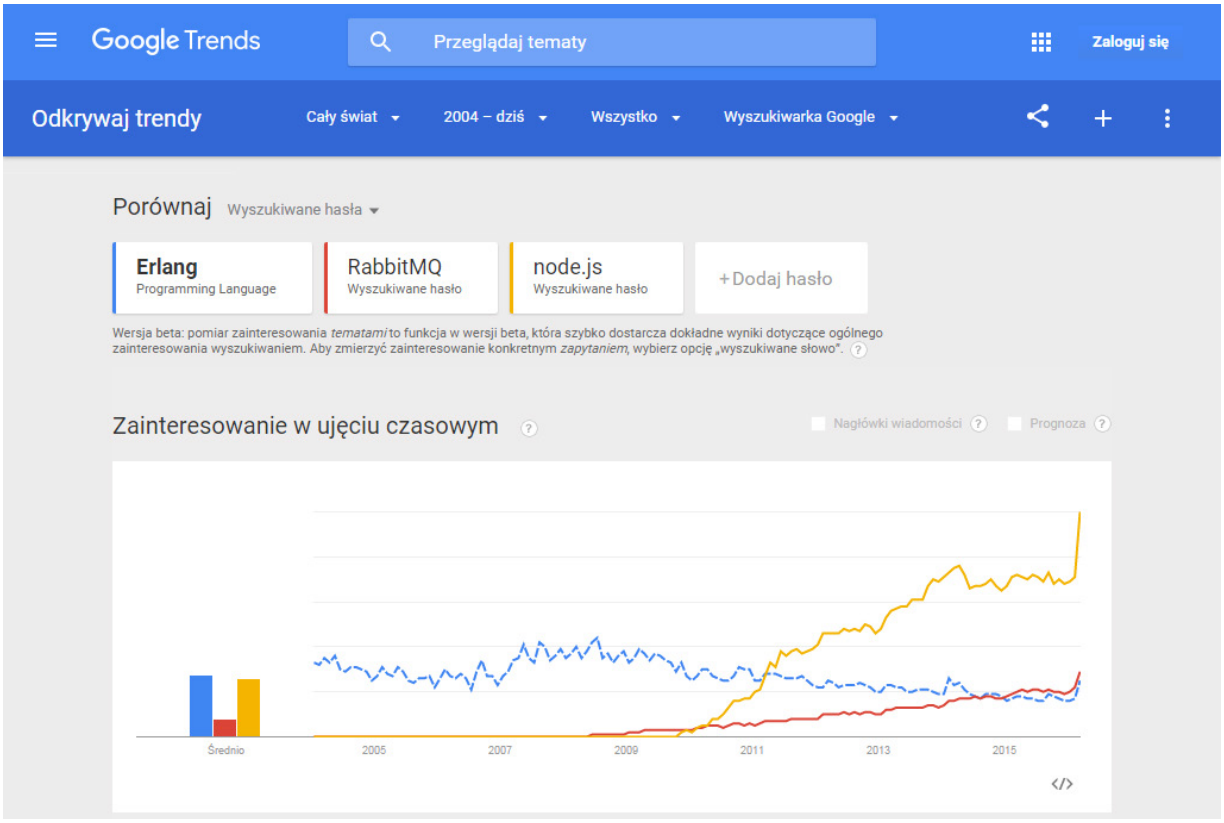
Erlang był własnościowym językiem programowania używanym przez Ericssona, jednak w roku 1998 kod źródłowy został otwarty, co przyczyniło się do popularyzacji tego języka wśród społeczności open source. Warto jednak zwrócić uwagę na rosnącą popularność systemu RabbitMQ na tle języka Erlang, co pokazuje Rysunek 1. Ze znajdującego się na nim wykresu wynika, iż język ten jest dojrzałym rozwiązaniem utrzymującym względnie stały stopień zainteresowania wśród specjalistów IT.

Zgodnie z przypuszczeniami – głównym zastosowaniem tego języka są systemy telekomunikacyjne (między innymi GPRS, 3G, LTE), jednakże wraz z popularyzacją systemów „rozproszonych" staje się on coraz popularniejszy także w innych sektorach IT.

Oto kilka projektów wykorzystujących język Erlang: Chef, CouchDB, WhatsApp, GitHub.

Język ten stosują w swoich rozwiązaniach między innymi następujące firmy: Amazon.com, Bet365, Facebook, Motorola, T-Mobile, Yahoo!

Na koniec warto wspomnieć, iż w pierwszym drukowanym numerze „Programisty" ze stycznia 2012 roku można znaleźć artykuł pod tytułem „Erlang - język inny niż C++ czy Java", którego lektura może pozwolić na bliższe przyjrzenie się jego funkcjom i możliwościom.



Rysunek 1. Popularność systemu RabbitMQ na tle języka Erlang i frameworka node.js w latach 2004-2016

WSTĘP DO SYSTEMU RABBITMQ

RabbitMQ jest projektem o otwartym kodzie źródłowym, napisanym w języku Erlang [3].

System ten jako pośrednik wiadomości (ang. *message broker*) natywnie implementuje protokół Advanced Message Queuing Protocol będący otwartym standardem opisującym protokół warstwy aplikacji dla oprogramowania pośredniczącego w wymianie wiadomości (ang. *Message-Oriented Middleware*, w skrócie MOM). Nasz kurier wiadomości obsługuje zarówno AMQP w wersjach 0-8, 0-9, 0-9-1 [4], jak i AMQP 1.0 [5], który znacząco różni się od wersji 0-9-1 i od którego jest bardziej rozbudowany.

RabbitMQ natywnie pracuje z protokołem AMPQ 0-8/0-9-*, zaś pozostałe protokoły są obsługiwane w formie pluginów.

Opisywany pośrednik wiadomości obsługuje również:

- protokół STOMP będący skrótem od Simple (lub Streaming) Text Oriented Message Protocol, znanym również jako TTMP, skoncentrowany na wymianie danych za pomocą tekstowego formatu (nie binarnego), który z powodzeniem może przeczytać człowiek;
- MQ Telemetry Transport (w skrócie MQTT) – lekki binarny protokół zdolny do operowania na niewielkich wiadomościach lub na łączach o niedużej przepustowości;
- HTTP – który jest wykorzystywany do łączności za pomocą JSON-RPC, WebSockets, SockJS, a także prostego HTTP API.

Protokół AMPQ 0-9-1

W niniejszym artykule znaleźć można przykłady wykorzystywania kolejek w oparciu o protokół AMPQ 0-9-1, który jest obsługiwany przez RabbitMQ natywnie, a więc bez pomocy dodatkowych pluginów, co ma miejsce w przypadkach innych protokołów, np. STOMP czy MQTT.

Słownik pojęć dotyczących AMPQ 0-9-1:

- Połączenie (ang. *connection*) jest połączeniem TCP, w którym AMQP jest protokołem w warstwie aplikacji. Jak długo połączenie jest utrzymywane, tak długo nadawca lub odbiorca będą mogli wysyłać bądź pobierać wiadomości z serwera.
- Kanał połączenia (ang. *channel*) jest wirtualnym bytem, który pozwala na dzielenie jednego połączenia TCP pomiędzy kilka kanałów, co pozwala na ich różnicowanie w obrębie jednej aplikacji – np. nadawania różnych wiadomości przez jednego nadawcę, które to mają trafić do kilku odbiorców.
- Wirtualny host (ang. *virtual host*) pozwalający tworzyć osobne środowiska z oddzielnymi uprawnieniami na bazie grup użytkowników, centrali wiadomości, kolejek itp.
- Kolejka wiadomości (ang. *message queue*) działająca zgodnie z zasadą FIFO (first in-first out).
- Centrala wiadomości (ang. *exchange*) określająca sposób filtrowania i/lub dyspozycji wiadomości na bazie określonych reguł.
- Klucz powiązania (ang. *routing key*) określający, czy do danej kolejki RabbitMQ ma wysłać otrzymaną wiadomość. Zastosowanie klucza routingu jest inne pomiędzy różnego rodzaju centralami wiadomości.
- Powiązanie centrali wiadomości z kolejką (ang. *binding*) na podstawie określonych reguł.

Większość cech tego protokołu opisano w dalszej części artykułu, jednak w tym momencie zostanie zaprezentowana podstawowa cecha tego protokołu, a więc budowa pojedynczej wiadomości (pakietu danych).

Ramka wiadomości
ładunek (ang. *payload*)
atrybuty

Tabela 1. Ramka wiadomości w protokole AMPQ 0-9-1