Techniczne aspekty niezawodnej chmury obliczeniowej

Paweł Lubomski, Paweł Pszczoliński, Andrzej Kalinowski

Politechnika Gdańska, Centrum Usług Informatycznych, ul. Narutowicza 11/12, 80-233 Gdańsk
{lubomski, pszczola, andrzej.kalinowski}@pg.gda.pl

Abstrakt

Powszechnie wykorzystuje się przetwarzanie w chmurze ze względu na redukcję kosztów zakupu sprzętu oraz łatwość zarządzania środowiskiem. W artykule zaproponowano architekturę prywatnej chmury obliczeniowej opracowanej i uruchomionej na Politechnice Gdańskiej, w ramach której wdrożono nowy portal MOST Wiedzy. Technologia wykorzystana do budowy zarówno chmury, jak i portalu wykorzystuje jedynie rozwiązania open source. Omówiono aspekty niezawodności takiego rozwiązania rozważając wysoką dostępność, łatwość utrzymania i wydajność. Wykorzystana architektura składająca się z bezstanowych usług typu REST jest elastyczna i pozwala na wytwarzanie zgodne z metodologią TDD (test-driven development), co wpływa na wyższą jakość oprogramowania. Programowanie funkcyjne umożliwiło sterowanie wykonaniem w sposób bardziej wydajnym niż z wykorzystaniem tradycyjnej w języku Java obsługi wyjątków. W ten sposób proces wytwarzania jest bardzo efektywny, a oprogramowanie wynikowe szybsze i wyższej jakości.

1. Wprowadzenie

Na przestrzeni ostatniej dekady dokonała się kolejna rewolucja technologiczna w projektowaniu architektur rozproszonych systemów informatycznych oraz świadczeniu usług hostingowych. Na porządku dziennym można zaobserwować wykorzystanie prywatnej, hybrydowej, a najczęściej publicznej chmury obliczeniowej [1]. Rozwiązanie to pozwoliło na zaoferowanie szerokiej gamy usług – od udostępnienia zwirtualizowanej infrastruktury (ang. *IaaS – Infrastructure-as-a-Service*), przez całe platformy systemowe (ang. *PaaS – Platform-as-a-Service*), po usługi skierowane do użytkownika końcowego (ang. *SaaS – Software-as-a-Service*). Wykorzystanie wielopoziomowej wirtualizacji [2] w znacznym stopniu zredukowało koszty zakupu i utrzymania infrastruktury sprzętowej. Dzięki modelowi płatności za wykorzystaną moc obliczeniową i przestrzeń dyskową (ang. *pay-as-you-go*) możliwe i opłacalne jest wynajęcie zasobów na krótkie przedziały czasu, np. na okres wzmożonego obciążenia. Sam proces zarządzania takim zwirtualizowanym sprzętem również uległ znaczącemu uproszczeniu. Dodatkowo zastosowanie konteneryzacji gotowych usług, np. z wykorzystaniem najpopularniejszego Dockera [3], sprowadził problem instalacji nowej usługi do kilku kliknięć w interfejs. Graficzne interfejsy administracyjne dostępne przez przeglądarkę WWW pozwalają na wygodny monitoring oraz zmiany w konfiguracji.

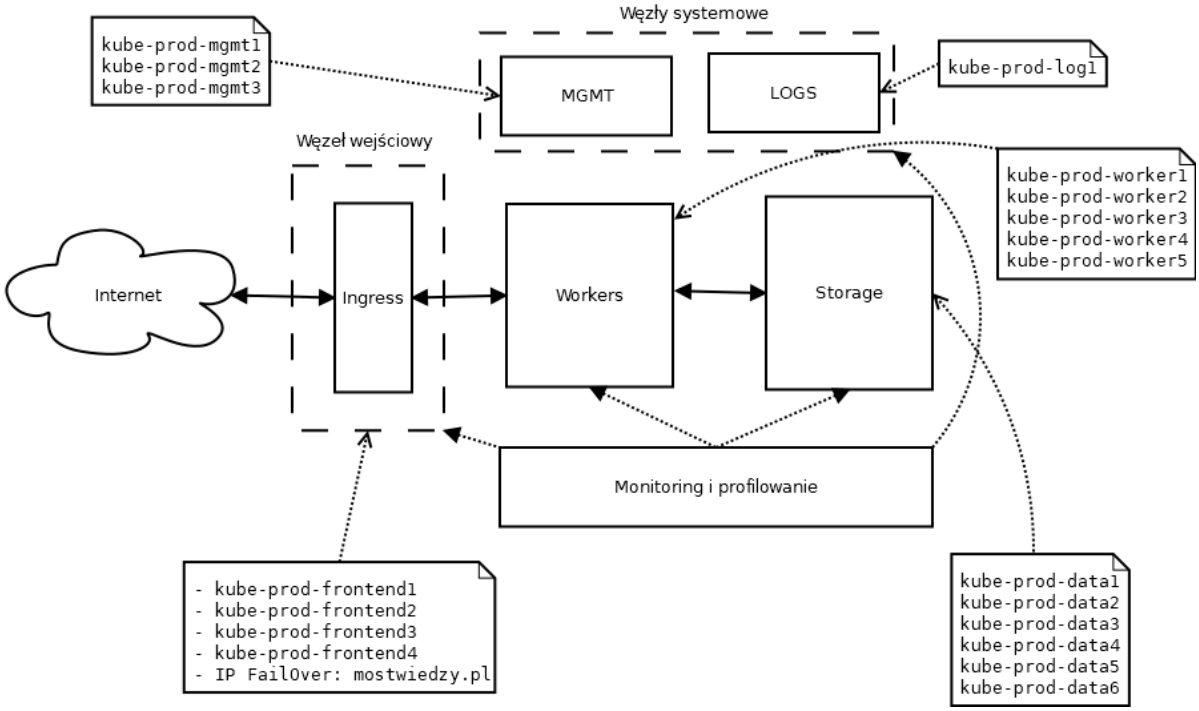
Prostota zarządzania i stosunkowo niewielki koszt kolejnych instancji tej samej usługi pozwoliły na projektowanie redundantnych, sklastrowanych usług i replikowanych zbiorów danych. Takie podejście wpłynęło na wzrost wiarygodności systemów – stały się one z punktu widzenia użytkownika niezawodne, o wysokiej dostępności. Dzięki zwielokrotnieniu węzłów obliczeniowych systemy charakteryzują się wydajnością pozwalającą obsłużyć bardzo duży wolumen równoczesnych połączeń i operacji [4].

Wykorzystanie wszystkich wspomnianych wyżej zalet wymaga jednak zmiany podejścia do projektowania systemów informatycznych. Nowoczesne systemy internetowe projektowane są jako bezstanowe usługi osiągalne przez protokół REST [5]. Można je w łatwy sposób rozpraszać i powielać. Wymaga również odmiennego podejścia do organizacji danych w bazach. Coraz częściej są to bazy danych NoSQL typu dokumentowego [6]. Tak fundamentalne zmiany w architekturze wymagają również odpowiedniego podejścia do zapewniania jakości wytwarzanego oprogramowania. W tych celach bardzo dobrze sprawdza się metodologia TDD (ang. *Test-Driven-Development*) [7].

2. Architektura i technologia prywatnej chmury obliczeniowej

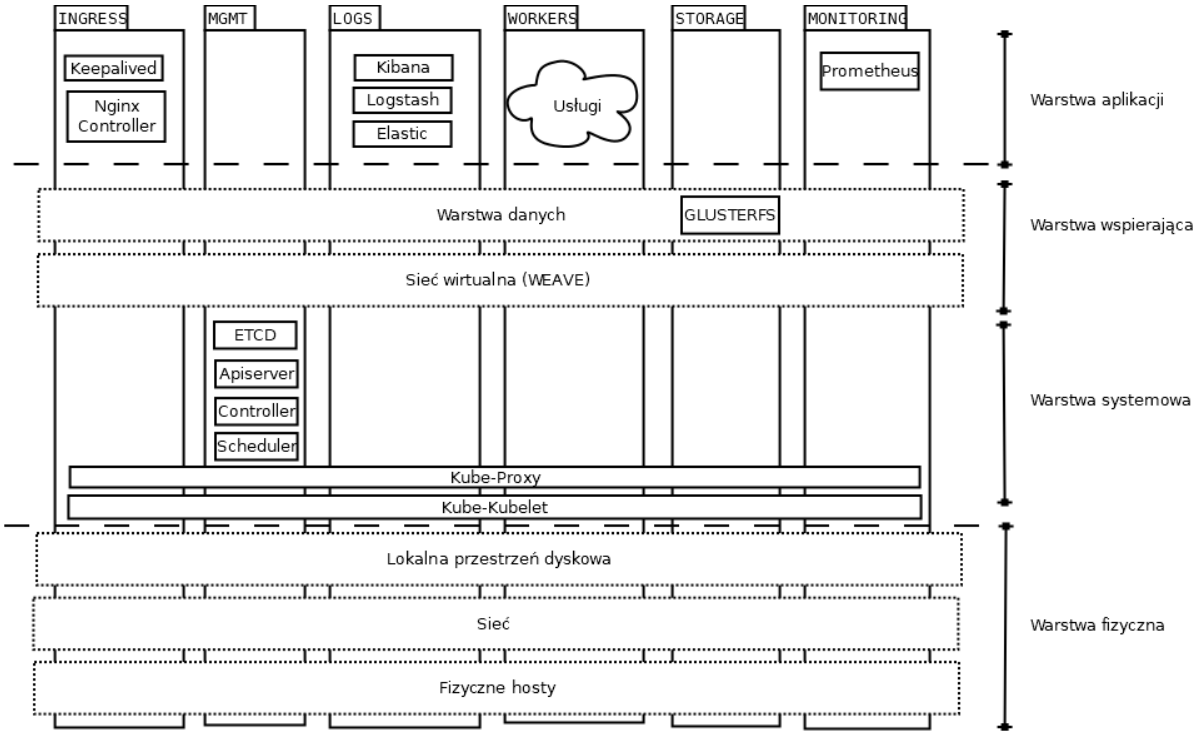
Politechnika Gdańska od lipca 2016 realizuje projekt „Multidyscyplinarny Otwarty System Transferu Wiedzy – MOST Wiedzy". Jest on współfinansowany jest z Europejskiego Funduszu Rozwoju Regionalnego w ramach Programu Operacyjnego Polska Cyfrowa na lata 2014-2020. W ramach projektu opracowano prywatną chmurę obliczeniową zlokalizowaną w centralnym węźle sieciowym Uczelni. Na tej chmurze uruchomiona została platforma MOST Wiedzy [8], która korzystając z wielu baz danych gromadzi informacje o różnych przejawach działalności naukowo-badawczej pracowników Uczelni, a następnie udostępnia je promując ich dorobek i osiągnięcia. Stanowi ona również instytucjonalne repozytorium pełnych treści publikacji dostępnych bezpłatnie na licencjach Creative Commons [9].

Opracowana architektura chmury obliczeniowej wykorzystuje tylko technologie dostępne na licencjach open source. Rys. 1 przedstawia ogólną architekturę rozwiązania.



Rys. 1. Ogólna architektura prywatnej chmury obliczeniowej Politechniki Gdańskiej

Poszczególne węzły fizyczne, na których uruchamiane jest środowisko, posiadają charakterystykę sprzętową dostosowaną do roli, jaką pełnią. Na węzłach wejściowych uruchamiana jest usług Nginx Ingress Controller [10], która przechwytuje ruch przychodzący i przekazuje go do odpowiednich węzłów obliczeniowych wewnątrz chmury. Dodatkowo może również obsługiwać terminację SSL, dzięki czemu ruch wewnątrz chmury jest nieszyfrowany, co znacząco poprawia wydajność. Na węzłach wejściowych dodatkowo uruchomiona jest usługa odpowiedzialna za przełączanie adresów IP w przypadku awarii węzła obsługującego IP zewnętrzne (ang. *IP Failover*). Wszystkie węzły pracują pod kontrolą oprogramowania Kubernetes [3][11]. Dwa podstawowe rodzaje węzłów to węzły obliczeniowe (ang. *workers*) oraz przechowujące dane (ang. *storage*). Bardzo istotną rolę pełnią węzły zarządzające (ang. *management – mgmt*). To one kontrolują działanie wszystkich usług uruchomionych w chmurze oraz uruchamiają nowe instancje usługi w ramach potrzeby. Wykorzystują do tego oprogramowanie ETCD [12]. Ostatnim typem węzła jest węzeł agregujący logi ze wszystkich usług. Bez centralnego mechanizmu zbierania logów systemowych praktycznie niemożliwy byłoby monitoring logów tak rozproszonych i zreplikowanych usług.



Rys. 2. Model warstwowy prywatnej chmury obliczeniowej Politechniki Gdańskiej

Rys. 2 przedstawia model warstwowy opracowanego środowiska chmurowego z uwzględnieniem zastosowanych technologii i rozwiązań. Wyróżnić można 4 zasadnicze warstwy. Najniżej położona warstwa fizyczna odpowiadająca wykorzystanemu sprzętowi. Kolejna to warstwa systemowa – w niej uruchomione są kontenery dockerowe wymagane przez oprogramowanie Kubernetes do działania. Warstwa wsparcia, w której uruchomione są aplikacje wspierające działanie pozostałych usług, wykorzystuje GlusterFS [13] oraz wirtualną sieć Weave [14]. Ostatnia warstwa to warstwa aplikacji, w której uruchamiane są usługi chmurowe (tzw. PODy) [15].

Przeprowadzone testy i symulacje na bazie platformy MOST Wiedzy [8] wdrożonej na opisanej prywatnej chmurze obliczeniowej wykazały pełną odporność na awarię zarówno poszczególnych usług (żądania kierowane były do pozostałych), jak i całego węzła (wszystkie usługi zostały odtworzone na innych węzłach). Nawet w przypadku awarii węzła wejściowego, jego obowiązki natychmiast przejmuje inny węzeł wejściowy. W najbardziej niekorzystnym przypadku z punktu widzenia niektórych użytkowników niedostępność oscylowała na poziomie 1-2 sekund potrzebnych na detekcję awarii i przekierowanie ruchu do innej usługi bądź węzła. Skutkowało to utratą przeważnie 0-1 żądania, co jest w pełni akceptowalnym wynikiem w dobie systemów internetowych.

Wprowadzenie graficznego interfejsu zarządzającego z poziomu strony WWW pozwala w bardzo łatwy sposób monitorować i zarządzać usługą. Również skalowanie poziome (klastrowanie) jest proste i sprowadza się do zwiększenia liczebności usług poszczególnego typu. W ten sposób można zwiększać wydajność bez konieczności kupowania bardzo drogiego sprzętu (w przypadku skalowania pionowego).

3. Architektura systemu wykorzystująca usługi REST

Efektywne wykorzystanie chmury obliczeniowej wymaga wsparcia ze strony architektury systemu na niej wdrożonego. Wszystkie funkcjonalności muszą być podzielone na skalowalne usługi, które mogą być publikowane na wielu węzłach. Architektura mikroserwisów [16] jest w takiej sytuacji naturalnym rozwiązaniem. Należy jednak spełnić pewne wymagania, aby zapewnić efektywne i niezawodne wykorzystanie tej architektury. Usługi powinny być pogrupowane w logiczne spójne

kontrolery. Każdy kontroler powinien posiadać własny punkt dostępu o unikatowym adresie URL do metody REST.

Każda usługa jest bezstanowa i odpowiedzialna za jedną logiczną i atomową operację biznesową. Protokół REST z formatem JSON jest prosty, wydajny i może być używany przez dowolnego klienta. Jedną z wad takiej konfiguracji dostępu do usług jest brak dokumentacji. Interfejs REST API nie ma formalnego opisu sygnatur metod. Aby rozwiązać ten problem, potrzebna jest dodatkowa, automatyczna dokumentacja. Taka dokumentacja musi być łatwa do wygenerowania i uzupełnienia oraz musi zawierać opis struktur danych dla żądań i odpowiedzi. Dzięki dokumentacji nowi klienci będą mogli bezbłędnie wywoływać dostępne API usług.

Głównym celem protokołu REST jest oddzielenie klienta i usługi tak, aby mogły ewoluować oddzielnie. W związku z tym, że sam protokół komunikacyjny nie narzuca ograniczeń na przysyłane dane, walidacja ich poprawności musi być wykonana po stronie logiki biznesowej. Parametry żądania weryfikowane są z uwzględnieniem kontekstu i semantyki przesyłanych danych. Dodatkowo protokół ten ma zdecydowanie niższy narzut komunikacyjny w stosunku do protokołu SOAP.

Mając zapewnioną kontrolę jakie dane są odbierane przez usługi konieczne jest jeszcze sprawdzenie kto wysyła żądania. Bezpieczeństwo musi być zapewnione na dwóch poziomach. Pierwszy z nich – system musi sprawdzić, czy nadawca żądania jest zaufany i nie będzie możliwy nieautoryzowany dostęp. Drugi – system musi sprawdzić, czy ten zaufany nadawca ma prawa dostępu do tej wybranej usługi. Autoryzacja żądania powinna sprawdzać nie tylko czy autor żądania jest właściwy, ale także to czy żądanie jest wciąż aktualne oraz że nie została podmieniony żaden parametr żądania podczas przesyłania. Dlatego w każdym żądaniu należy wysłać identyfikator nadawcy, datę wysłania, sumę kontrolną generowaną z całego żądania oraz token. Tajny klucz do generacji tokenu powinien być dostarczony klientowi poza systemem i najlepiej regularnie zmieniany. W przypadku, gdyby ktoś przechwycił żądanie i chciał je wysłać jeszcze raz, to zabezpieczenie w postaci walidacji daty ważności tokenu nie pozwali na dostęp do usługi. Gdyby przechwycone żądanie zostało zmienione (data ważności zostałaby przedłużona lub zamieniony zostałby dowolny parametr żądania), to suma kontrolna nie będzie się zgadzać i nie będzie możliwe wykonanie usługi, ponieważ bez tajnego klucza nie można wygenerować prawidłowej sumy kontrolnej.

Po przyznaniu autoryzowanego dostępu role powinny być przypisane do usług i klientów. Gdy aplikacja klienta ma odpowiednią rolę, może wykonać wybraną usługę. Oczywiście poziom gradacji może różnić się w zależności od potrzeb. Można stosować dwie role – odczytu i zapisu, a z drugiej strony możliwe jest zdefiniowanie po jednej roli dla każdej istniejącej usługi. W celu poprawy efektywności zarządzania rolami można je połączyć w grupy ról, a klientów przypisywać do całych grup zamiast pojedynczych ról.

Z punktu widzenia biznesu powyższe wymagania są konieczne i bez nich nie da się zapewnić niezawodności i wysokiej wydajności. Jakość tworzonych usług opiera się również na dwóch zagadnieniach, o które należy zadbać. Pierwszym z nich są testy automatyczne, które powinny sprawdzać nie tylko algorytmy biznesowe każdej z usług, ale również wszystkie właściwości usługi, o których mowa powyżej. Oczywiście konieczne są testy jednostkowe do sprawdzania poprawności wykonywanych algorytmów. Testy integracyjne powinny być wykorzystywane do sprawdzania operacji na bazie danych czy plikach lub operacji komunikacyjnych.

Testy kontrolerów REST sprawdzają właściwości żądania i walidacje. Testy bezpieczeństwa sprawdzają autoryzację i kontrolę dostępu, a testy obciążeniowe poprawną wydajność i skalowalność. Zapewnienie czasu na tworzenie testów podczas procesu wytwarzania pozwala zaoszczędzić czas w trakcie okresu utrzymania usług. Takie podejście w naturalny sposób wskazuje na wykorzystanie metodologii TDD (ang. *test-driven development*) [7]. Tworzenie oprogramowania w metodologii TDD wymaga rozpoczęcia pracy od pisania testów dla danego fragmentu przyszłej implementacji rozwiązania. Następnie implementacja jest realizowana tak długo, dopóki wszystkie testy nie zostaną poprawnie wykonane. Obecnie często stosuje się też modyfikację TDD, która zakłada jednoczesne pisanie implementacji i testów, tak aby nowy kod funkcjonalny powstawał szybko i jednocześnie zapewniona była jego wysoka jakość.

Drugim bardzo ważnym zagadnieniem, który oszczędza wiele czasu w okresie utrzymania usług jest logowanie zdarzeń. Dobrze zdefiniowane dzienniki z wykonania usług są niezbędne dla systemu produkcyjnego działającego w środowisku chmury. Jeśli usługi działają na wielu węzłach asynchronicznie, identyfikacja błędu jest możliwa jedynie dzięki poprawnej sekwencji wpisów w dzienniku logów. Dlatego logowanie do plików lub bazy danych powinno być nieodłączną częścią kodu usług.

Ostatnim aspektem o którym warto wspomnieć przy tworzeniu nowoczesnego systemu jest wykorzystanie programowania funkcyjnego [17]. Dzięki wykorzystaniu możliwości jakie oferuje programowanie funkcyjne można znacznie uprościć kod systemu oraz zwiększyć wydajność wykonania dzięki łatwemu wykorzystaniu wielowątkowości przy przetwarzaniu złożonych zagadnień.

Powyższe wymagania są realizowane podczas tworzenia systemu MOST Wiedzy, żeby w pełni wykorzystać zalety chmury obliczeniowej oraz zapewnić produkt najwyższej jakości oparty na najnowszych rozwiązaniach technologicznych.

4. Podsumowanie

MOST Wiedzy to nowoczesny system internetowy budowany w oparciu o najnowsze rozwiązania dostępne w świecie open-source. Wykorzystanie prywatnej chmury obliczeniowej umożliwia skuteczne wykorzystanie zasobów sprzętowych przy jednoczesnym zapewnieniu wydajności i niezawodności działania. Aby chmura obliczeniowa mogła wspierać w pełni pracę systemu jest on tworzony w oparciu o bezstanowe usługi typu REST. Wśród zastosowanych rozwiązań znajdują się wszystkie niezbędne, aby system MOST Wiedzy był skalowalny, bezpieczny, dobrze udokumentowany, łatwy w utrzymaniu i wydajny. Wykorzystanie najlepszych wzorców projektowych i metodologii wsparte wykorzystaniem dedykowanych bibliotek i platform gwarantuje wytwarzanie produktu najwyższej jakości.

Bibliografia:

[1] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: State-of-the-art and research challenges,” *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[2] P. Lubomski, A. Kalinowski, and H. Krawczyk, “Multi-level Virtualization and Its Impact on System Performance in Cloud Computing,” in *Communications in Computer and Information Science*, vol. 608, 2016, pp. 247–259.

[3] D. Bernstein, “Containers and Cloud: From LXC to Docker to Kubernetes,” *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[4] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, “Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads,” in *2010 IEEE 3rd International Conference on Cloud Computing*, 2010, pp. 228–235.

[5] L. Richardson and S. Ruby, *RESTful Web Services*. 2008.

[6] J. Han, E. Haihong, G. Le, and J. Du, “Survey on NoSQL database,” in *Proceedings - 2011 6th International Conference on Pervasive Computing and Applications, ICPCA 2011*, 2011, pp. 363–366.

[7] S. Hammond and D. Umphress, “Test driven development: the state of the practice,” *Proceedings of the 50th Annual Southeast Regional Conference*, pp. 158–163, 2012.

[8] “Multidyscyplinarny Otwarty System Transferu Wiedzy - MOST Wiedzy.” [Online]. Available: http://mostwiedzy.pl/. [Accessed: 10-Feb-2017].

[9] “Creative Commons Licenses,” 2015. [Online]. Available: http://creativecommons.org/licenses/.

[10] “Nginx Ingress Controller.” [Online]. Available: https://github.com/kubernetes/ingress/tree/master/controllers/nginx. [Accessed: 12-May-2017].

[11] “Production-Grade Container Orchestration.” [Online]. Available: https://kubernetes.io/. [Accessed: 12-May-2017].

[12] “Operating etcd clusters for Kubernetes.” [Online]. Available: https://kubernetes.io/docs/concepts/cluster-administration/configure-etcd/. [Accessed: 12-May-2017].

[13] “GlusterFS - Storage for your Cloud.” [Online]. Available: https://www.gluster.org. [Accessed: 23-Aug-2017].

[14] “Weave Cloud: Simplify Containers & Microservices.” [Online]. Available: https://www.weave.works/. [Accessed: 23-Aug-2017].

[15] “Pods.” [Online]. Available: https://kubernetes.io/docs/concepts/workloads/pods/pod/. [Accessed: 23-Aug-2017].

[16] A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture,” *IEEE Software*, vol. 33, no. 3, pp. 42–52, May 2016.

[17] V. Subramaniam, “Functional Programming in Java: Harnessing the Power of Java 8 Lambda Expressions,” *Harnessing the Power of Java 8 Lambda Expressions*, p. 171, 2014.

Artykuł powstał w ramach projektu “Multidyscyplinarny Otwarty System Transferu Wiedzy – MOST Wiedzy” – umowa nr POPC.02.03.01-00-0014/16-00.