JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 26 (2023) 63–67
Received: 26 September 2022
Accepted: 7 November 2022

Analysis of configuration distribution methods in service application environments

Analiza metod dystrybucji konfiguracji w środowiskach aplikacji usługowych

Arkadiusz Bryczek*, Piotr Kopniak

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
This work is an analysis of methods and solutions enabling centralization of the storage of configuration values in service application environments. It presents a comparison focused on the user's interaction with the available solutions and performance tests during the operation. Consul, ZooKeeper tools and the implementation of the configuration server with the use of the Spring Cloud Config library were tested. Created test environment with the use of Java Spring, Kafka and Python technologies was used for this purpose. The aim of the research is to determine the best tool for working in large microservice networks and the most optimal in the context of working with the user. The presented results confirm the advantage of the Consul tool in terms of efficiency and interface quality over other solutions.

Keywords: configuration distribution; microservices; cloud solutions

Streszczenie
Niniejsza praca jest analizą metod oraz rozwiązań umożliwiających centralizację przechowywania wartości konfiguracyjnych w środowiskach aplikacji usługowych. Prezentuje ona porównanie ukierunkowane na współpracę użytkownika z dostępnymi rozwiązaniami oraz badanie wydajność podszas wykonywania operacji. Testom poddano narzędzia Consul, ZooKeeper oraz implementację serwera konfiguracyjnego z wykorzystniem biblioteki Spring Cloud Config. Wykorzystane do tego celu zostało autorskie środowisko testowe stworzone z użyciem technologii Java Spring, Kafka oraz Python. Celem jest wyznaczenie najlepszego z narzędzi do pracy w dużych sieciach mikroserwisowych oraz najoptymalniejszego w kontekście pracy z użytkownikiem. Zaprezentowane wyniki potwierdzają przewagę narzędzia Consul w aspekcie wydajności oraz jakości interfejsu nad pozostałymi rozwiązaniami.

Słowa kluczowe: dystrybucja konfiguracji; mikroserwisy; rozwiązania chmurowe

*Corresponding author
Email address: **arkadiusz.bryczek@pollub.edu.pl** (A. Bryczek)

1. Wstęp

Rozwiązania chmurowe zmieniają sposób rozwoju i wdrażania aplikacji ze względu na ich unikalne wymagania. Deweloperzy dążą do uzyskania wyższej skalowalności i niezawodności. Dynamiczne środowiska jakimi są środowiska chmurowe mogą wymagać posiadania magazynu przechowującego konfigurację [1]. Autonomiczny charakter nieodłącznie związany z mikrousługami wymaga od twórcy aplikacji wzięcia odpowiedzialności za aspekty operacyjne, takie jak łatwa konfigurowalność [2]. W przypadku architektury monolitycznej, gdzie posiadamy jedną główną aplikację, a co za tym idzie jeden plik przechowujący wartości konfiguracji nie jest wymagane wydzielanie jej do wspólnego miejsca. Posiadając wiele mikroserwisów koniecznym byłoby powielenie tych zasobów, a w momencie edycji, wykonanie jej w wielu miejscach dla części wspólnych. Pomocne w tej sytuacji stają się narzędzia, które umożliwiają aplikacjom na jednym środowisku pobranie konfiguracji po HTTP. Wybór odpowiedniego narzędzia staje się ważnym elementem analizy w tworzeniu architektury systemu.

Technologie do zarządzania konfiguracją służą do przechowywania wartości konfiguracyjnych i umożliwiają dynamiczną zmianę wartości. Budując środowisko zastosowanie technologii przechowujących konfigurację takich jak Consul lub ZooKeeper to powszechna praktyka podczas tworzenia sieci mikroserwisów [2].

Celem artykułu jest przeprowadzenie i prezentacja wyników badań określających wydajność oraz jakość interfejsu narzędzi Consul, ZooKeeper oraz implementacji serwera konfiguracyjnego z wykorzystaniem biblioteki Spring Cloud Config, które wzbogacają aplikację budowaną w architekturze mikroserwisowej o scentralizowany magazyn przechowujący konfigurację oraz umożliwiający jej szybką podmianę. Zważając na open source'ową dystrybucje narzędzia Consul, łatwość konfiguracji oraz wbudowany graficzny interfejs można wysunąć tezę, że charakteryzuje go lepsza wydajność od pozostałych badanych aplikacji oraz jakościowo lepszy interfejs.

Rozdział drugi przybliży badany temat oraz źródła wiedzy wykorzystanej do stworzenia prezentowanej pracy. W rozdziale trzecim zaprezentowane zostanie

środowisko przygotowane do przeprowadzenia badań. Metody badawcze oraz opis scenariuszy testowych zostanie zawarty w rozdziale czwartym. Rozdział piąty przedstawi wyniki otrzymane jako rezultat przeprowadzonych testów oraz ich omówienie. Dyskusja i porównanie zebranych danych z uprzednio publikowanymi badaniami oraz dostępnymi źródłami zostaną zawarte w rozdziale szóstym. Siódmy rozdział zawierał będzie podsumowanie oraz wnioski zebrane w czasie eksperymentu.

2. Przegląd literatury

W artykule Omar'a Al.-Debagy'ego i Peter'a Martinek'a [3] przedstawiono porównanie architektury monolitycznej z architekturą mikroserwisową. Udowodniono, że zastosowanie podejścia budowy sieci aplikacji składającej się z wielu współpracujących serwisów jest bardziej optymalne w aspekcie użytkowania przez większą liczbę użytkowników. Korzyści są również widoczne na płaszczyźnie ładowania aplikacji, szybkości restartu w przypadku błędu. Zbadany został również czas odkrywania aplikacji w środowiskach usługowych, potwierdzono tym wysoką wydajność oraz wsparcie w komunikacji między serwisami oraz to, że architektura mikroserwisowa nie utrudnia połączenia i zależnej współpracy aplikacji.

Artykuł [4] dotyczy ewolucji narzędzia ZooKeeper oraz zawiera opis możliwej do realizacji metody dynamicznego przeładowywania konfiguracji. Wykorzystany w badaniu został schemat pracy, w którym aplikacja kliencka zamiast regularnego odpytywania serwera o zmiany ustawia na obszary, które chce monitorować obserwatorów. Zadaniem ich jest nasłuchiwanie na zdarzenia w subskrybowanych gałęziach, do których ZooKeeper wysyła powiadomienia w momencie aktualizacji kluczy konfiguracji. W momencie, gdy klient odbierze takie powiadomienie, wysyła zapytanie do aplikacji w celu pobrania aktualnych danych oraz przeładowuje swoje ustawienia. Podejście to odstępuje od tradycyjnego, lecz pozwala to ograniczyć obciążenie poprzez reakcję aplikacji klienckiej tylko w sytuacji, gdy zmiana została dokonana w interesującym go obszarze. Optymalizuje to obciążenie i szybkość pracy w przypadku wysokiej liczby podpiętych klientów.

W artykule autorstwa Andisheh Feizi oraz Chui Yin Wong [9] zawarte zostały badania porównujące współpracę użytkowników z interfejsami graficznymi oraz wierszem poleceń. Testom poddana została efektywność pracy osób początkujących oraz wdrożonych w użytkowanie danego oprogramowania. Interfejs graficzny był postrzegany jako łatwiejszy do nauczenia dla obu grup. Potwierdzone zostało, że praca z nim jest efektywniejsza, jeśli jest poprawnie zbudowany. Pod uwagę wzięte zostały aspekty takie jak odpowiednia treść etykiet, odpowiednio dobrane ikony odnoszące się do zawartości zakładek oraz głębokość, oznaczająca ilość podstron dla danej strony. Stwierdzono również, że najlepszym rozwiązaniem jest udostępnienie graficznego interfejsu o możliwościach analogicznych do interfejsu konsolowego.

Badania dotyczące komunikacji narzędzi do dystrybucji konfiguracji takich jak ZooKeeper lub Consul z klientami, uwzględniające blokowanie połączenia przez każdego klienta oraz zwalnianie go zostały uwzględnione w artykule Piotra Grzesika i Dariusza Mrozka [5]. Został tam zbadane czasy uzyskiwania przez klienta blokady podczas połączenia oraz zwalniania jej w określonych środowiskach oraz pod obciążeniem. Badania pokazują, że narzędzia wypadają niekorzystnie w przypadku, gdy kilku klientów próbuje uzyskać dostęp do tego samego klucza. Najlepszą wydajnością charakteryzuje się jednak ZooKeeper, który w najgorszym scenariuszu zaprezentował najlepszy średni czas blokady i zwolnienia.

Powyższe publikacje zawierają eksperymenty potwierdzające, że wykorzystanie magazynu konfiguracyjnego jako mikroserwisu nie wpływa negatywnie na ogólną wydajność całego systemu. Potwierdzone zostało również, że za pomocą jednego narzędzia można zastosować różne podejście w zbudowaniu dynamicznego systemu dystrybuującego wartości zdefiniowane w konfiguracji. Żadne z nich nie prezentuje badania wydajności narzędzi podczas pracy w dużej sieci mikroserwisów. Brakuje również porównania wydajności w scenariuszach, gdy aplikacja jest powiadamiana o zmianie wartości w magazynie, a regularnym odpytywaniu magazynu o stan przechowywanych w nim wartości. Badaniu nie została też poddana współpraca użytkownika z narzędziem poprzez udostępniony interfejs.

3. Środowisko badawcze

Bazą do przeprowadzenia badania są wspomniane narzędzia do przechowywania konfiguracji oraz aplikacja testowa napisana w języku Java z wykorzystaniem szkieletu programistycznego Spring Boot oraz jego integracji z rozwiązaniami chmurowymi [6]. W aplikacji testowej wykorzystane zostaną gotowe biblioteki Spring Cloud oraz zmienne umożliwiające uzyskanie połączenia z aktualnie badanym narzędziem do przechowywania konfiguracji oraz kolejką wiadomości Kafka opisaną w dalszej części rozdziału. Aplikacja zawierać będzie jedną klasę posiadającą pole, pod które będą wczytywane wartość z pliku konfiguracyjnego przechowywanego w magazynie. W momencie odebrania informacji o zmianie wartości klucza, aplikacja zapisze czas zdarzenia oraz prześle go do tymczasowego magazynu danych. Dodatkowo utworzona została w języku Python aplikacja pomocnicza zapisującą czas zatwierdzenia zmian w konfiguracji.

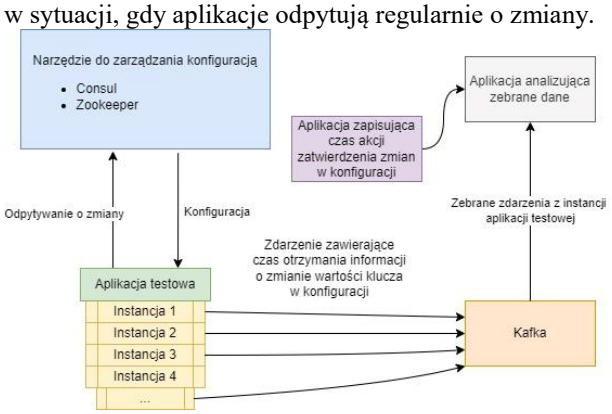
Osobną częścią będzie konfiguracja kolejki do strumieniowego przesyłania wiadomości Kafka [7]. Posłuży ona jako magazyn tymczasowo przechowujący zebrane z instancji aplikacji testowej czasy, które będzie wysyłała aplikacja w momencie, gdy zostanie odebrana informacja o zmienionej wartości klucza w konfiguracji.

Równie dobrym rozwiązaniem byłby zapis do bazy danych, umożliwiający przechowywanie otrzymanych wyników oraz przechowywanie ich. Obrane podejście wykorzystania kolejki pozwala na bieżącą analizę danych i monitorowanie testu podczas jego przebiegu.

Ostatnią częścią jest aplikacja analizująca zebrane w kolejce zdarzenia. Skonfigurowane zostanie połączenie oraz nasłuchiwanie na zdarzenia w kolejce. Serwis będzie przyjmował dane z kolejki oraz zwracał je jako rezultat zapytania.

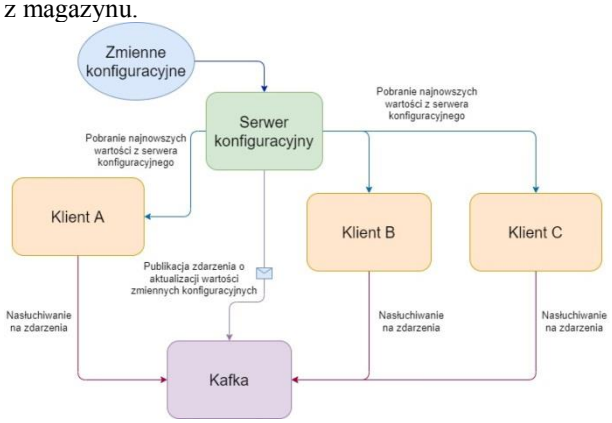
Uruchomienie środowiska zostanie zrealizowane z wykorzystaniem platformy konteneryzacji Docker. Każda z aplikacji uruchomiona zostanie jako osobny kontener pomijając aplikację pomocniczą, która będzie uruchamiana przed każdym testem. Pomocna w badaniu będzie funkcjonalność skalowania kontenerów. Dzięki niej w prosty sposób za pomocą flagi „scale" powielone zostaną instancje aplikacji testowej. Pozwoli to zachować równość aplikacji w środowisku związaną z przydzieleniem zasobów oraz poprawi rzetelność badań.

Na Rysunku 1 przedstawiono połączenia oraz informacje o danych przekazywanych pomiędzy poszczególnymi aplikacjami w utworzonym środowisku w sytuacji, gdy aplikacje odpytują regularnie o zmiany.



Rysunek 1: Połączenia w środowisku testowym – odpytywanie przez aplikacje o zmiany.

Zawarty na Rysunku 2 diagram zawiera sieć komunikacyjną aplikacji w sytuacji, gdy instancje aplikacji testowej są powiadamiane o zmianach, a następnie wykonywane jest pobranie danych z magazynu.



Rysunek 2: Połączenia w środowisku testowym – poinformowanie aplikacji o wprowadzonych zmianach.

4. Metody badawcze

Zaprezentowane środowisko posłuży do przeprowadzenia dwóch typów badań. Wykonane zostanie obiektywne badanie wydajność metod oraz subiektywne badanie interfejsu przeprowadzone na grupie testowej składającej się z dziesięciu studentów informatyki.

Analiza wydajności zostanie wykonana poprzez zbadanie czasu odebrania informacji o wprowadzeniu zmian oraz reakcji na to zdarzenie w dwudziestu sześciu instancjach aplikacji testowej. Wszystkie zostaną podpięte do jednej instancji narzędzia do przechowywania konfiguracji. Na wydajność składa się zarówno szybkość reakcji badanego narzędzia oraz jego optymalizacja w aspektach obciążenia systemu. W badanej technologii dokonana zostanie zmiana wartości klucza, która zapoczątkuje przeładunek we wszystkich instancjach zaimplementowanego rozwiązania. Każda z instancji w momencie uzyskania informacji o zdarzeniu zmiany wartości klucza zapisze pod zmienną dokładny co do wartości tysiącznych sekundy czas. Następnie wszystkie instancje prześlą zapisaną wartość do kolejki, która posłuży jako tymczasowy magazyn. W tle uruchomiona będzie aplikacja nasłuchująca na dane wpływające do tego magazynu. Dla większej dokładności badania, test zostanie powtórzony dziesięć razy dla każdej z metod. Obliczenia oparte będą o zebrane dane czasowe oraz czas startu badania zapisanego przez wspomnianą aplikację pomocniczą.

Na podstawie otrzymanych wiadomości zostanie obliczony średni czas przeładunku konfiguracji we wszystkich instancjach aplikacji testowej dla danej metody. Analogiczny scenariusz zostanie wykonany dla każdego z badanych rozwiązań.

Badanie interfejsu użytkownika zostanie przeprowadzone na grupie dziesięciu osób, które wcześniej nie miały kontaktu z żadnym z badanych narzędzi. Każdy badany otrzyma przed rozpoczęciem adekwatną dla narzędzia instrukcję opisującą krok po kroku w jaki sposób odnaleźć wartość klucza, którą będzie miał za zadanie zmienić. Użytkownik zapoznaje się z instrukcją, a następnie deklaruje gotowość do odbycia badania. Każda badana osoba wykona osobne scenariusze dla każdego z narzędzi.

Grupa dokona również oceny interfejsu poprzez indywidualne wypełnienie ankiety określającej jakość interfejsu na podstawie ankiety LUT (ang. Lublin University of Technology) [8]. Wybrane zostały w tym celu z ankiety obszary:

- nawigacja i struktura składająca się z podobszarów: łatwość nawigowania, hierarchia informacji oraz struktura informacji,
- komunikaty, pomoc dla użytkownika uwzględnia elementy takie jak: komunikaty ogólne, komunikaty o błędach oraz informacje zwrotne i pomoc,
- interfejs aplikacji obejmujący dwa podobszary: layout oraz dobór barw,

- wprowadzanie danych, który składa się z podobszarów odnoszących się do formularzy oraz wprowadzanych danych.

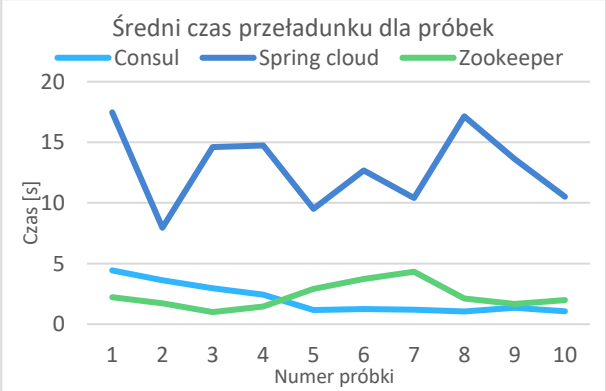
Każda z sekcji zawiera pytania oraz odpowiedzi w postaci oceny określanej jako wartość liczbowa z przedziału od jeden do pięć, gdzie jeden oznacza słabe wsparcie danego aspektu w interfejsie badanego narzędzia, a pięć jego łatwość i intuicyjność oraz prostotę w wykorzystaniu. Rezultatem obliczenie współczynnika WUP (ang. Web Usability Points) [8]. Wykorzystany zostanie do tego wzór:

$$WUP = \frac{1}{n_a}\sum_{i=1}^{n_a}\frac{1}{s_i}\sum_{j=1}^{s_i}\frac{1}{q_{i_j}}\sum_{k}^{q_{i_j}} p_{ijk} \quad (1)$$

gdzie n_a oznacza liczbę obszarów w ankiecie, S_i liczbę podobszarów w obszarze i, q_{i_j} liczbę pytań w obszarze i oraz podobszarze j a p_{ijk} to ocena przyznania przez osobę badaną pytaniu o numerze k w podobszarze j obszaru i.

5. Wyniki

Badania wydajności pozwoliły zapisać czas od startu do zakończenia operacji przeładowania w każdej z instancji aplikacji testowej. Na Rysunku 3 przedstawiono wykres średnich czasów przeładowania każdej próbki dla każdej badanej metody. Można zaobserwować na nim, że Consul oraz ZooKeeper wykazały podobne wyniki oraz rozbieżności zebranych czasów.



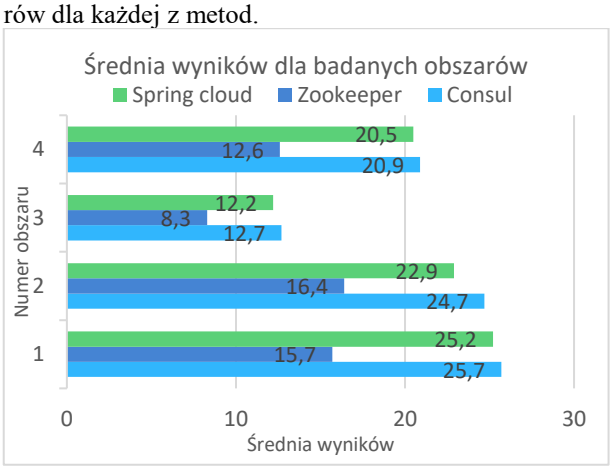
Rysunek 3: Średni czas przeładunku dla próbek.

Obliczając średnią z czasów próbek dla każdej z metod wyznaczono średni czas przeładunku oraz wartość rozbieżności dla danych.

Tabela 1: Wyniki czasowe badania wydajności metod

Metoda	Czas[s]	Odchylenie standardowe
Consul	2.056	1,233
ZooKeeper	2.314	1,043
Spring Cloud Config Server	12.865	3,223

Badanie jakości interfejsu wykonane przy pomocy ankiety LUT[8] skutkowało zebraniem zestawu danych na podstawie, którego obliczony został współczynnik jakości interfejsu WUP. Rysunek 4 przedstawia wykres średniej wyników dla każdego z poszczególnych obszarów dla każdej z metod.



Rysunek 4: Średnia wyników dla badanych obszarów.

W Tabeli 2 zawarte zostały średnie z obliczonych dla każdej badanej osoby współczynników, dla każdej z badanych metod. Uwzględniona została również rozbieżność otrzymanych wyników.

Tabela 2: Wyniki badania jakości interfejsu

Metoda	WUP	Odchylenie standardowe
Consul	4.23	0.53
ZooKeeper	2.68	0.22
Spring Cloud Config Server	4.03	0.38

6. Dyskusja

Celem przeprowadzonych badań było potwierdzenie tezy o przewadze narzędzia Consul nad podobnymi mu metodami tj. narzędziem ZooKeeper oraz implementacją serwera konfiguracyjnego z wykorzystaniem biblioteki Spring Cloud.

Wyniki badań wydajnościowych jednoznacznie wskazują na przewagę narzędzia oraz zastosowania metody, w której aplikacja pobiera informację o zmianach. Magazyny konfiguracyjne Consul oraz ZooKeeper uzyskały zbliżone wyniki w badaniu czasu reakcji, gdzie nieznaczną przewagą uzyskał serwer konfiguracyjny Consul. Obie aplikacje były notyfikowane o zmianach w ten sam sposób. Potwierdza to, że podejście, gdzie aplikacja odpytuje o zmiany serwer konfiguracyjny jest optymalniejsze oraz bardziej wydajne. Czas reakcji aplikacji oraz czas przeładowania wartości konfiguracyjnych jest krótszy, więc cała operacja zostanie również wykonana szybciej we wszystkich podpiętych pod serwer serwisach. Wyniki dla implementacji serwera konfiguracyjnego są znacząco gorsze. Jest to spowodowane zastosowanym mechanizmem notyfikacji, który mógłby okazać się lepszy, gdyby zwiększyć skalę testu. Podobne wyniki zaprezentowane zostały w artykule [5],

gdzie podobnie porównane zostały oba narzędzia w kontekście uzyskiwania klucza blokady oraz rywalizacji o klucz. Obie aplikacje podobnie wykazały nieznacznie różne wyniki, lecz w tym przypadku wydajniejszy był ZooKeeper.

Wyniki badania jakości interfejsu użytkownika otrzymane dzięki wynikom ankiety oraz policzonemu na ich postawie współczynnikowi zaprezentowały, że narzędziem posiadającym najlepszy jakościowo interfejs jest Consul. Analizując zebrane wyniki zaobserwować można przewagę interfejsów graficznych nad interfejsami konsolowymi. Użytkownicy podobnie wskazali, że interfejsy, gdzie użytkownik przeprowadza operacje poruszając się oraz wybierając elementy widoczne na ekranie są bardziej intuicyjne oraz łatwiejsze w użytkowaniu. Artykuł [9] prezentuje podobne wyniki potwierdzając, że praca z interfejsem graficznym aplikacji charakteryzuje się większą efektywnością w aspekcie pracy zarówno dla osoby początkującej oraz posiadającej doświadczenie w pracy z narzędziem.

7. Wnioski

Analizując wyniki przeprowadzonych badań można jednoznacznie stwierdzić, że w obydwu przypadkach najlepszymi wynikami charakteryzuje się Consul. Narzędzie uzyskało czas lepszy o 258 milisekund od serwera konfiguracyjnego ZooKeeper. Różnica jest nieznaczna, lecz skala testu zakładała przeładunek konfiguracji w 26 instancjach aplikacji testowe. Większa skala testu mogłaby spowodować uzyskanie większej różnicy w wynikach dla badanych serwerów konfiguracyjnych. Najgorszy wynik uzyskała implementacja serwera konfiguracyjnego z wykorzystaniem biblioteki Spring Cloud Config. Serwer charakteryzuje się czasem reakcji ponad 10 sekund większym od pozostałych badanych narzędzi. Związane jest to z innym mechanizmem powiadamiania, który charakteryzuje się mniejszym obciążeniem ze względu na brak konieczności ciągłego odpytywania o zmiany. Z drugiej strony zastosowanie kolejki wiadomości jako pośrednika do powiadamiania aplikacji o wprowadzonych zmianach znacznie wpływa na czas reakcji aplikacji.

Badania jakości interfejsu wypadły dla narzędzia Consul równie korzystnie. W każdym z obszarów aplikacja otrzymywała najwyższe wyniki. Współczynnik WUP wyniósł dla niego 4,23 co jest wynikiem o 0,2 lepszym od implementacji serwera konfiguracyjnego. Biblioteka zakłada wykorzystanie systemu kontroli wersji Git, jako interfejsu. W przypadku narzędzia Consul interfejs przystosowany edycji wartości konfiguracyjnych, poprzez przeznaczony do tego ekran. Wpływa to na większą intuicyjność oraz czytelność, której brakuje w systemach kontroli wersji ze względu na ich odmienne przeznaczenie. Najgorsze wyniki w tej sekcji otrzymał ZooKeeper, którego współczynnik WUP jest o 1,35 niższy od serwera Spring Cloud Config. Nie posiada wbudowanego graficznego interfejsu przez co praca z nim w kontekście scenariuszy okazała się najtrudniejsza, a jego interfejs najmniej intuicyjny. Trudności sprawiało użytkownikom również wyszukiwanie informacji w menu pomocy zawierającego polecenia wraz z opisem parametrów przez nie wykorzystywanych.

Literatura

[1] J. Wu, T. Wang, Research and Application of SOA and Cloud Computing Model, 2014 Enterprise Systems Conference, Shanghai (2014) 294-299, http://doi.org/10.1109/ES.2014.58.

[2] S. Kehrer, W. Blochinger, AUTOGENIC: Automated generation of self-configuring microservices, Department of Computer Science, Reutlingen University (2018) 35-46, http://dx.doi.org/10.5220/0006659800350046.

[3] O. Al-Debagy, P. Martinek, A comparative review of microservices and monolithic architectures, IEEE 18th International Symposium on Computational Intelligence and Informatics (2018) 149-154, http://dx.doi.org/10.1109/CINTI.2018.8928192.

[4] C. M. Pham, Z. Kalbarczyk, R. K. Iyer, V. Dogaru, R. Wagle, C. Venkatramani, An evaluation of zookeeper for high availability in system S, Proceedings of the 5th ACM/SPEC international conference on Performance engineering (2014) 209-217, http://doi.org/10.1145/2568088.2576801.

[5] P. Grzesik, D. Mrozek, Evaluation of key-value stores for distributed locking purposes, Beyond Databases, Architectures and Structures, Silesian University of Technology (2019) 70-81, https://doi.org/10.1007/978-3-030-19093-4_6 .

[6] Dokumentacja Spring Cloud w integracjach z innymi rozwiązaniami chmurowymi, https://spring.io/projects/spring-cloud, [30.01.2022].

[7] Dokumentacja Apache Kafka, https://kafka.apache.org/, [30.01.2022].

[8] M. Miłosz, Ergonomia systemów informatycznych, Lublin University of Technology, Lublin, 2014.

[9] A. Feizi, C. Y. Wong, Usability of user interface styles for learning a graphical software application, Faculty of Creative Multimedia, Malaysia (2012) 1089-1094, http://dx.doi.org/10.1109/ICCISci.2012.6297188.