УДК 621.396.1

ОСОБЛИВОСТІ ТЕСТУВАННЯ СИСТЕМ З МІКРОСЕРВІСНОЮ АРХІТЕКТУРОЮ

Остапов О.А., Шевченко І.І., Чуб М.М.
Інститут телекомунікаційних систем КПІ ім.Ігоря Сікорського, Україна
E-mail: chub.mykhailo@lll.kpi.ua

Features of testing systems with microservice architecture

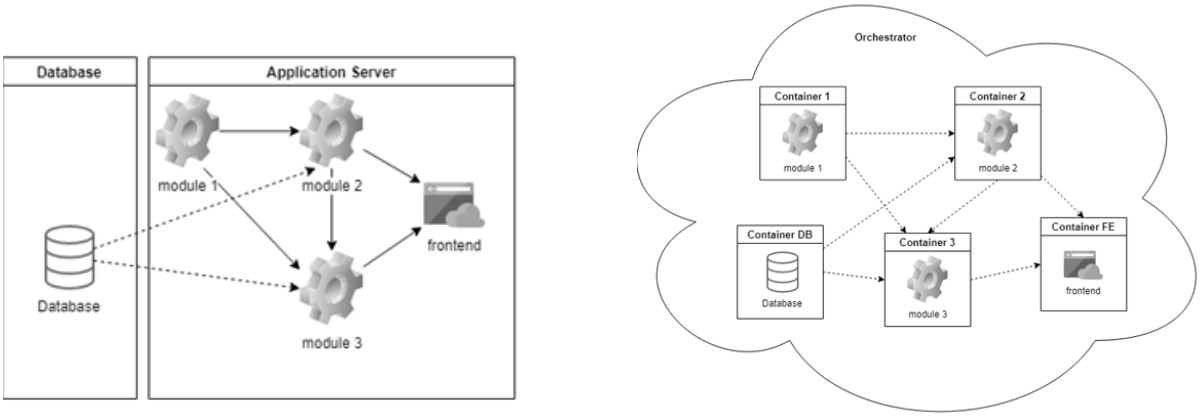
The current trend of creating web applications is based on the use of microservice architecture. The paper describes difficulties of testing microservices.

Одним з передових підходів створення інформаційних систем є використання мікросервісної архітектури інформаційної системи. Особливістю мікросервісної архітектури є те, що кожен сервіс розгортається незалежно від інших і реалізує виконання певної заданої функціональності системи, а отже данні зберігаються децентралізовано. Крім того, при використанні даного підходу для досягнення бізнес-потреб немає обмежень на вибір технології для реалізації кожного мікросервісу. Отже, мікросервіси - це декомпозицією великої бізнес-системи на незалежні елементи, які легко розгорнути і легко обслуговувати, при цьому кожен з них виконує одну певну задачу[1,2].

Типовий життєвий цикл розробки програмного забезпечення складається з наступних фаз: збір і аналіз вимог (Requirement analysis), дизайн (Design) реалізація (Implementation), тестування (Testing), розгортання (Deployment) та підтримка (Maintenance). Тестування відповідає за забезпечення якісної роботи продукту, супроводжується перевіркою працездатності системи, виявленням, фіксацією і виправленням помилок з метою досягнення необхідних стандартів якості. Метою даної статті є огляд основних особливостей тестування систем з мікросервісною архітектурою по відношенню до систем з монолітною. [3,4]

Концепція монолітного програмного забезпечення розглядається як традиційний метод розробки додатків та полягає в тому, що різні компоненти програми об'єднуються в одну програму на одній платформі, а отже всі модулі, що відповідають за різну бізнес-логіку, вміщено в одній програмі та запущено в одному процесі.

Мікросервіси надають змогу виконувати ту ж саму роботу, що й моноліт, але розподіливши модулі в окремі віртуальні машини-контейнери за принципом «один модуль — одне завдання — один контейнер» і зв’язавши їх за допомогою віртуальної мережі [5].



а) б)

Рис.1. Архітектура програмного забезпечення: а) монолітна, б) мікросервісна.

Мікросервіси складно тестувати атомарно. В якості програмних інтерфейсів для взаємодії з мікросервісами, як правило, використовують RESTful API і асинхронний обмін повідомленнями через черги. При тестування веб-сервісів з визначеними наборами даних для монолітної архітектури достатньо локально розгорнути систему. У 99% випадків не можна запустити один окремий мікросервіс, щоб протестувати його REST веб-сервіси, наприклад, не створивши спочатку Mock-server всіх пов'язаних з ним сервісів, що унеможливлює інтеграційне тестування.

Кожен мікросервіс може мати свою окрему базу даних, ніяк не пов'язану з базами інших мікросервісів. З погляду автономності даний підхід є раціональним, але якщо об'єкти в системі, що тестуються, мають дочірні об'єкти кожного мікросервісу, база даних не може забезпечити їхньої цілісності. Рішенням є створення окремого механізму для зв'язності даних у кожному мікросервісі, наприклад, зберігати ID об'єктів інших сервісів. Проблема в тому, що дані одного мікросервісу можуть бути стерті, а помилку можна знайти лише на етапі тестування. Рішенням даної проблеми є створення та редагування даних через UI в невеликих обсягах під кожний конкретний тест.

Для мікросервісів важко забезпечити транзакційність. Тестування транзакцій у мікросервісах полягає в тому, що перший сервіс робить зміни в даних і передає їх другому сервісу — другий сервіс робить зміни у своїх даних і передає третьому. Під час такого тестування може виникнути помилка, наприклад, необробленого Null Pointer Exception. Тест не пройшов, дані перших двох мікросервісів не відповідають іншим — потрібно вручну їх чистити. Для забезпечення транзакційності між різними мікросервісами виникає необхідність розробникам писати більше коду, більше веб-сервісів для зворотного зв'язку, що потребує більше часу на тестування. З погляду цілої системи може бути зовсім не очевидно, в якому саме мікросервісі сталася

помилка, для чого необхідно вручну перевіряти Log системи та данні всіх залучених сервісів. У монолітах же транзакційність гарантують на рівні бази даних.

При монолітній архітектурі ПЗ може містити певну кількість веб-сервісів, створених для виконання бізнес-функцій програми. При мікросервісній архітектурі кількість веб-сервісів значно збільшується, оскільки, крім тих самих сервісів для виконання бізнес-функцій, додаються технічні сервіси для зв'язку мікросервісів між собою, які також потребують тестування. Кожен веб-сервіс одночасно повинен передавати більше даних (рис 2), а кожний новий канал комунікації ускладнює тестування й часто потребує встановлення спеціальних програм для роботи.

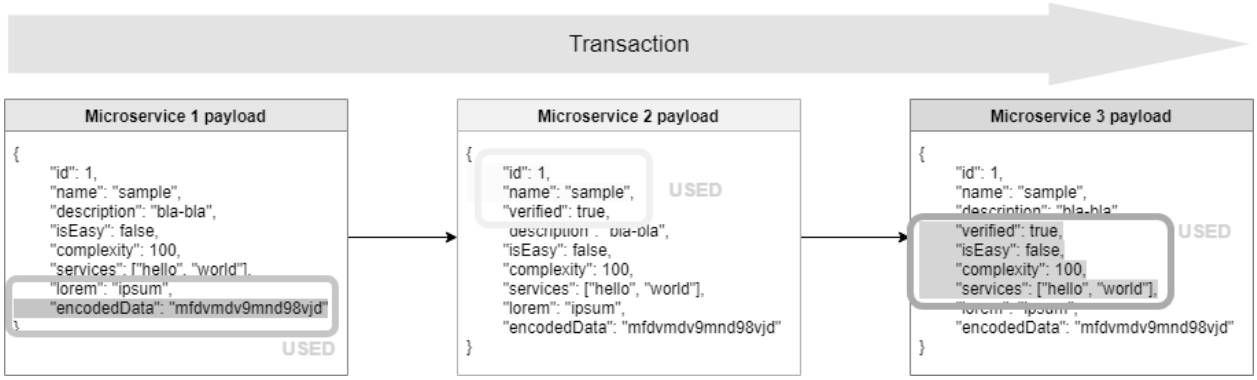


Рис.2. Передача даних при мікросервісній архітектурі.

Мікросервіси потребують значної кількості ресурсів, адже кожен контейнер мікросервісу - власна віртуальна машина, що використовує ресурси. У разі ж монолітної архітектури є лише один сервер чи віртуальна машина.

В роботі описано основні особливості та основні підходи до тестування програмного забезпечення з мікросервісною архітектурою.

Література

1. Ільченко М.Ю., Кравчук С.О. Телекомунікаційні системи. – К.: Наукова думка, 2017.
4. nergy efficiency analysis of hybrid-ARQ relay-assisted schemes in LTE-based systems nergy efficiency analysis of hybrid-ARQ relay-assisted schemes in LTE-based systems.
2. Кравчук С.О., Шевченко І.І., Чуб М.М. Технології, що можуть прискорити розвиток інфокомунікаційної галузі та становлення інформаційного суспільства// Матер. 13-ї міжнар. наук.-техн. конф. "Перспективи телекомунікацій", 15-19 квітня 2019 р. – К.: КПІ ім. Ігоря Сікорського, 2019. – С. 192–195.
3. Rajasekharaiah C. Microservices, Here and Beyond. In: Cloud-Based Microservices. Apress, Berkeley, CA., 2021 https://doi.org/10.1007/978-1-4842-6564-2_10.
4. Rao, P. V. , V. P.Kumar, and B. P. K.Reddy . 2018. "Applying Agile Software Methodology for the Development of Software Development Life Cycle Process (Sdlc)." Journal for Research— Volume 4 (2): 125–136.
5. G. Campeanu, "A mapping study on microservice architectures of internet of things and cloud computing solutions", 2018 7th Mediterranean Conference on Embedded Computing (MECO), pp. 1-4, June 2018.