УДК 004.4

Марченко О.О., магістрант, група ЗІСТ-19м
Вакалюк Т.А., д-р. пед. наук., доц, професор кафедри
Державний університет «Житомирська політехніка»

ПЕРЕВАГИ ТА НЕДОЛІКИ ВИКОРИСТАННЯ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ ПРИ РОЗРОБЦІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

За останнє десятиріччя розробка програмного забезпечення на основі мікросервісної архітектури стала одним із головних трендів у сфері інформаційних технологій. Поява і розвиток мікросервісів пов'язана з еволюцією в області розробки програмного забезпечення в цілому. Упродовж тривалого часу використовувався підхід, орієнтований на використання монолітних додатків. Ці додатки розроблялися однією командою і розгорталися як єдине ціле. У певний момент стало очевидним, що в ряді випадків розбиття моноліту на декілька незалежних додатків може принести перевагу. Цей підхід стали активно використовувати і просувати великі компанії, такі як, наприклад, Netflix, eBay, Google та інші.

Під мікросервісом зазвичай розуміють незалежну програму, що виконує чітко окреслений набір функцій і взаємодіє з іншими програмами через певний інтерфейс та за допомогою загальновизнаних, стандартних та легких протоколів, наприклад HTTP. Ключова ідея мікросервіса – незалежність від середовища, слабка з ним зв'язаність і високий рівень повторного використання. Мікросервісна архітектура має ряд суттєвих переваг. Згадаємо деякі з них:

– Підвищення загальної швидкодії та продуктивності сервісу за рахунок асинхронної обробки запитів.

– Можливість використання різних мов програмування та способів збереження даних. Наприклад, один сервіс може бути написаний на мові PHP, а для іншого сервісу, який вимагає більшої швидкодії, може бути використаний Golang або C++. Що ж до способів збереження даних, то, наприклад, для збереження записів або коментарів у чаті може бути використане документо-орієнтоване noSQL сховище, для системи авторизації – реляційна база даних, для збереження користувацького обміну повідомленнями у соціальній мережі – графова база даних, а для зберігання медіафайлів – сховище blob-об'єктів.

– Відмовостійкість: при виході з ладу одного або ж навіть декількох сервісів зберігається загальна працездатність усієї системи. У цьому випадку також значно легше ізолювати причину проблеми.

– У великих монолітних системах доводиться розширювати всю систему одразу. При роботі ж із невеликими сервісами можна розширювати лише ті з них, які цього потребують, і дають можливість запускати інші частини системи на менш потужному обладнанні. Також за допомогою системи балансування можна згладжувати пікові навантаження на окремі сервіси.

– У системах-монолітах внесення змін навіть в один рядок коду може потягнути за собою потребу зміни у багатьох частинах програми, яка може складатися з десятків і навіть сотень тисяч рядків коду. Натомість, при використанні мікросервісної архітектури можна вносити зміни в окремий сервіс та значно швидше публікувати його незалежно від всього проєкту. Це означає, що нові функціональні можливості зможуть дійти до користувачів значно швидше. Разом з цим у випадку виявлення некоректної роботи нового функціоналу можна швидко цю проблему виявити та ізолювати або ж, навіть, відкотити зміни.

Але слід враховувати і певні недоліки такої архітектури:

– Додаються накладні витрати на комунікацію між сервісами та базою даних. У випадку нестабільної мережі або великої кількості запитів – це може стати проблемою. Разом із цим потрібно добре продумати протоколи і стандарти обміну інформацією між сервісами.

– Якщо у випадку монолітної архітектури ми маємо супроводжувати одну програму, то у випадку мікросервісів потрібно підтримувати багато програм і, часто, серверів. Із цього також випливає потреба у більш кваліфікованому персоналі.

– Не слід вважати, що перехід на мікросервісну архітектуру вирішує проблему чистоти коду. Якщо код написаний непрофесійно, то при розділенні проєкту на N мікросервісів, ми отримаємо N непрофесійно написаних сервісів.

Отже, можемо зробити висновок, що мікросервісна архітектура не є єдино правильним шляхом вирішення усіх проблем на конкретному проєкті. Це лише один з інструментів у руках програміста, який потрібно використовувати з розумом і у тих випадках, коли це реально підвищує швидкодію програми, робить код більш зрозумілим і, таким чином, оптимізує зусилля на розробку нового функціоналу і підтримку раніше написаного. Зростання популярності такої схеми побудови програмного забезпечення викликане зростанням рівня поширеності систем хмарних обчислень та потреби обробки значно більших обсягів інформації та, як наслідок, зростанням трафіку і загального навантаження на сервери. Тому кожен сучасний програміст має володіти цією технологією.