Кравчук О.А., Синюк О.М.

Хмельницький національний університет

Переваги та недоліки мікросервісної архітектури

Загальноприйнятою практикою розробки програм була розробка за принципом монолітної архітектури, яка передбачає проект як одну програму, яка відповідає за весь необхідний функціонал. І, безумовно, такий підхід мав свої переваги:

простота розробки - IDE (середовище розробки) та інші інструменти для створення програмного забезпечення сфокусовані на побудові проекту як одного цілого;

легкість тестування - проект легше тестується, коли весь функціонал знаходиться в одній програмі;

простота розгортання – проект легко розгорнути, оскільки він складається з одного файлу (наприклад, WAR, JAR тощо).

Але, при розвитку та збільшенні проекту, монолітна архітектура розпочинає вносити певні недоліки в процес розробки:

– додавання певного нового функціоналу супроводжується наростанням кодової бази;

– швидкість розробки сповільнюється через необхідність внесення змін в одну велику систему;

– дотримання границь відповідальності внутрішніх модулів розмивається;

– будь-яка зміна проекту вимагає його перекомпіляції;

– компіляція, збирання та розгортання програми займає більше часу;

– розуміння проекту та його архітектури ускладнюються через збільшення його розмірів, новим розробникам потрібно більше часу для інтеграції.

Враховуючи вище наведені недоліки монолітної архітектури, розробники почали використовувати інші підходи. Одним із них є мікросервісна архітектура.

Мікросервісна архітектура – варіант сервісної архітектури (модульний підхід ро розробки програм) програмного забезпечення, орієнтований на взаємодію максимально, наскільки це можливо, невеликих, слабо пов’язаних і

легко замінних модулів – мікросервісів [1]. При такій архітектурі проект являє собою набір невеликих сервісів, комунікація між якими відбувається за допомогою легких механізмів (наприклад, HTTP) [2]. Кожен з таких сервісів повинен виконувати свою чітко визначену бізнес-задачу.

Серед переваг такої архітектури можна виділити наступні [3]:

– незалежна розробка – невеликі за розмірами незалежні компоненти можуть створюватися малими незалежними командами розробників. Група розробників може працювати над змінами/розробкою одного сервісу, не змінюючи і навіть не знаючи про інший сервіс. Кількість часу, необхідна на вивчення сервісу значно зменшується, і реалізовувати нові функції стає значно простіше;

– незалежне розгортання – кожен окремий компонент проекту можна запускати незалежно від інших. Це дозволяє випускати новий функціонал швидше і з меншими ризиками. Виправляючи помилки в одному сервісі, його можна перезапускати без необхідності перезапускати інші сервіси;

– незалежне масштабування – кожен сервіс можна масштабувати незалежно від іншого. Наприклад, випускаючи новий функціонал, для збільшеного навантаження на сервери можна відмасштабувати лише сервіси, які будуть використовуватися користувачами активніше. Це дозволяє зменшити витрати та виконувати масштабування гнучкіше, ніж при монолітній архітектурі. Також, для кожного сервісу можна підібрати найкращу конфігурацію ресурсів (пам’яті, процесора тощо), в той час як монолітна система буде працювати на одному комп’ютері;

– можливість повторного використання – кожен компонент реалізує свою чітко визначену бізнес-задачу. За рахунок цього, кожен сервіс можна використовувати в інших проектах, виконуючи менше змін для адаптації;

– краща ізоляція неполадок – в мікросервісній архітектурі краще ізольовані неполадки. Наприклад, витік пам’яті в одного сервіса вплине лише на нього, в той час як інші сервіси продовжать працювати в звичайному режимі. В монолітній архітектурі така проблема виведе з роботи всю систему;

– краща здатність до залучення нових технологій – кожен сервіс може використовувати інший набір технологій, іншу мову розробки тощо. Якщо в випадку невдалого використання технології в мікросервісній архітектурі необхідно змінити лише компоненти, які використовують цю технологію, в монолітній системі необхідно буде переписувати всю програму.

Однак, при всіх позитивних моментах, мікросервісна архітектура має також недоліки:

– збільшена складність для розробників – якщо розробнику необхідно виконати завдання, яке пов’язане з декількома сервісами, зазвичай це збільшує складність розробки, оскільки йому необхідно буде конфігурувати та запускати ці декілька сервісів на своєму комп’ютері, що є важчим ніж запуск однієї програми;

– складність розподілених систем – у випадку взаємодії декількох сервісів необхідно бути готовим до часткових збоїв, високої латентності та недоступності інших віддалених сервісів;

– складність тестування – зростає складність у написанні автоматичних інтеграційних тестів, оскільки система складається з незалежних між собою сервісів;

– зростання складності експлуатації – команді, яка займається експлуатацією та підтриманням сервісів у робочому стані буде важче виконувати свої обов’язки. Замість однієї програми необхідно буде контролювати певну кількість сервісів, і чим їх більше, тим більша кількість потенційних проблемних місць;

– необхідна серйозна компетентність – розробка проекту за допомогою мікросервісної архітектури може дати продуктивні результати, якщо роботу виконували професіонали. Однак, маючи низький рівень освіченості, розробка таких проектів може привести до заплутаності в організації сервісів та неефективності результату роботи;

– необхідність встановлення чітких границь – для кожного сервісу повинні бути визначені чіткі границі його бізнес-задачі. Якщо цього не буде зроблено, тоді залежності між сервісами будуть зростати, що може привести до того, що

через ці самі залежності необхідно буде запускати певні сервіси як одну єдину групу, де один сервіс не може функціонувати без іншого. В такому випадку систему можна назвати розподіленим монолітом.

Література

1. Chris Richardson: Microservices Patterns: With examples in Java, Manning Publications; 1 edition (November 19, 2018), ISBN: 9781617294549
2. Мартін Фаулер, Джеймс Льюїс – Мікросервіси [Електронний ресурс]. – Режим доступу : https://martinfowler.com/articles/microservices.html
3. Стівен Уоттс, Лора Шифф - Огляд архітектури моноліту проти мікросервісів [Електронний ресурс]. – Режим доступу : https://www.bmc.com/blogs/microservices-architecture/