УДК 621.395

МЕТОД ПОБУДОВИ ЗАВАДОСТІЙКОГО ПРОГРАМНОГО КОМПЛЕКСУ З ВИКОРИСТАННЯМ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

Маньківський В.Б., Омельченко С. А.
Навчально-науковий інститут телекомунікаційних систем
КПІ ім. Ігоря Сікорського, Україна
E-mail: v.b.mankivskiy@gmail.com , llamemnonal@gmail.com

THE METHOD OF BUILDING A FAULT-TOLERANT MICROSERVICE SOFTWARE COMPLEX

This article describes the principle of operation and construction of a software system using microservice architecture, as well as the risks and benefits. Comparison of monolithic and microservice architecture. The algorithm for obtaining a noise-resistant software system for uninterrupted operation of the service is described.

У роботі описано принцип роботи та побудову програмного комплексу з використанням мікросервісної архітектури, вказані ризики та переваги. Порівняно монолітну та мікросервісну архітектуру. Описано алгоритм отримання завадостійкого програмного комплексу для безперебійної роботи сервісу.

Архітектура мікросервісів переносить логіку додатків у сервіси та використовує мережевий рівень для зв'язку між ними. Спілкування в мережі замість викликів у пам'яті призводить до додаткової затримки та складності системи, що вимагає взаємодії між кількома фізичними та логічними компонентами. Підвищена складність розподіленої системи призводить до більш високої ймовірності відмови окремих мереж. В цьому полягає основний ризик мікросервісної архітектури. При розробці мікрослужби завжди корисно подумати про те, як вона поводитиметься, якщо певний компонент не працює. Це допоможе створити відмовостійкий сервіс. Відмовостійкість (англ. "fault tolerance") це властивість, що дає змогу системі продовжувати нормально працювати в разі виходу з ладу деяких її компонентів.

Необхідно переконатися, що послуги не сильно залежать від одного компонента. І якщо вони виявляться залежними необхідно розробити стратегію швидкого відновлення після збою. В цьому і полягає різниця між монолітною архітектурою та мікросервісною архітектурою, у підходах до розбиття програмного забезпечення на компоненти та їх взаємодії. Адже у монолітній архітектурі весь програмний код та функціональні модулі розміщені в одному монолітному додатку. У цій архітектурі всі компоненти додатку взаємодіють між собою напряму, тому виникає проблема залежності компонентів один від одного. Це може призводити до складнощів з масштабуванням, розширенням додатку, а також зі збільшенням складності управління додатком та ризику виникнення

помилок. Тому для побудови завадостійкого програмного комплексу краще використовувати мікросервісну архітектуру.

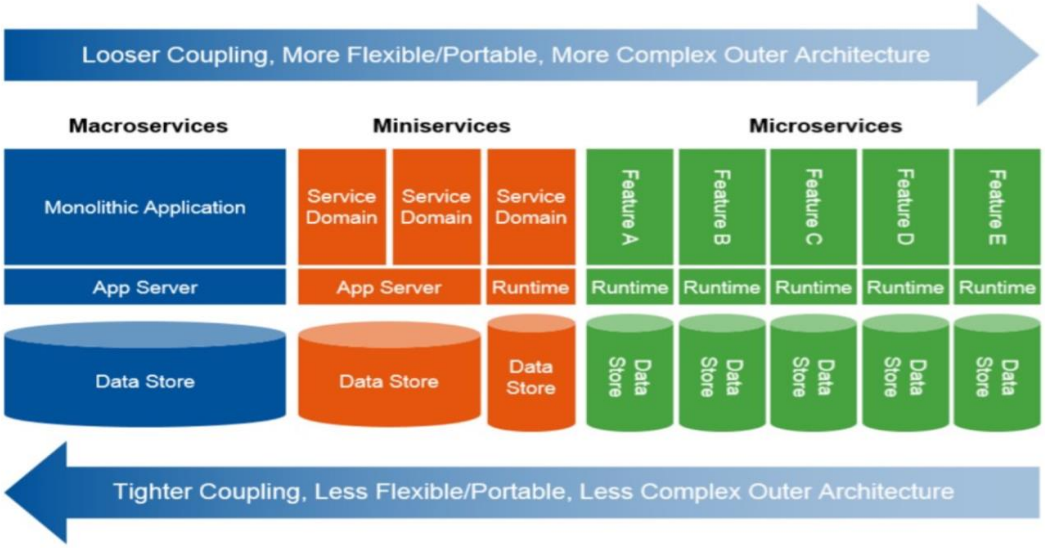

Рис. 1. Приклад сервісних архітектур.

Один із сучасних підходів у розробці інформаційних систем передбачає використання систем контейнеризації додатків для створення ізольованого оточення.

Першим плюсом цього підходу є незалежність оточення кожного додатка від встановлених на ЕОМ, що виконує контейнери, програм, що забезпечує прозорість і передбачуваність роботи додатка та захищає від перетину залежностей різних версій.

Другим плюсом є наявність механізмів управління та відстеження життєвого циклу контейнерів. Сучасні системи управління контейнерами (Docker Swarm, OpenShift , Kubernetes, Nomad) дають змогу контролювати стан програми всередині контейнера через настроюваний механізм перевірок і перезапускати неробочі контейнери.

Третім плюсом є декларативна система конфігурування розгортання програми. Під час розгортання потрібно вказати нову конфігурацію, і оркестратор приведе систему до потрібного стану, що дає змогу уникнути помилок під час імперативного написання команд. Важливим плюсом є можливість запускати оркестратор контейнерів на декількох ЕОМ, з'єднаних спільною мережею. Використання оркестратора в режимі декількох вузлів забезпечує безперебійну роботу контейнерів і їх переміщення на вільні вузли в разі відмови одного з них.

Окрім переваг, необхідно пам’ятати і про недоліки. Основний недолік мікросервісної архітектури – це продуктивність, скільки процеси проходять між сервісами, а от же в загальний час доданий час мережевої затримки між сервісами.

На відміну від монолітної архітектури (макросервісної), де лінії зв’язку – це внутрішня шина сервера. Це є основним обмеженням мікросервісної архітектури, яке потрібно розглядати, як особливість її і проектувати рішення виходячи з нього.

Отже, для побудови завадостійкого програмного комплексу з використанням мікросервісної архітектури необхідно виконати наступні кроки:

1. Аналіз вимог до системи та її функціональності.

2. Розробка незалежних мікросервісів: кожен мікросервіс повинен виконувати конкретну функцію, відповідальну за один або декілька зв'язаних між собою бізнес-процесів. Кожен мікросервіс повинен мати власну базу даних та інші ресурси, які не використовуються іншими мікросервісами.

3. Застосування контейнерів: Контейнери дозволяють ізолювати різні частини програмного комплексу, що сприяє зменшенню взаємодії між компонентами і забезпечує більшу стійкість системи до відмов.

4. Розробка API для мікросервісів: для взаємодії між мікросервісами необхідно розробити API, яке буде використовуватися для передачі даних та комунікації між мікросервісами.

5. Розробка механізмів відновлення: для забезпечення завадостійкості системи необхідно розробити механізми відновлення, які забезпечують автоматичне відновлення роботи системи в разі збоїв або помилок. Наприклад, механізми відновлення можуть включати автоматичне перенаправлення запитів до резервних мікросервісів, автоматичне відновлення бази даних або автоматичну перезавантаження мікросервісів.

6. Тестування та моніторинг: для забезпечення завадостійкості системи. Впровадження механізмів тестування та автоматизації розгортання та масштабування мікросервісів та встановлення системи моніторингу та контролю за роботою кожного мікросервісу і всього програмного комплексу в цілому. Це дозволить оперативно виявляти та вирішувати проблеми в системі.

Література

1. Gigi Sayfan. Hands-On Microservices with Kubernetes. Packt Publishing, 1st edition 2019, - p.p. 58-90.
2. Stephen Fleming. Microservices Architecture Handbook: Non-Programmer's Guide for Building Microservices. CreateSpace Independent Publishing Platform 2018, - p.p. 62-73.
3. P.S. Kocher. Docker microservices and containers, Addison-Wesley Professional, 1st edition 2019, - p.p. 189 -201.
4. Orkhan Gasimov. Microservice architecture for beginners.
https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/
5. Newman S. Creating microservices, 2016, - p.p. 124-149.
6. M. Fowler. Enterprise software application architecture, Addison-Wesley Professional, 1st edition 2006, p. 544.
7. Neal Ford. Building Microservice Architectures, 2015, p. 20.