УДК 004

ПЕРЕВАГИ ТА НЕДОЛІКИ ЗАСТОСУВАННЯ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ НА ПЛАТФОРМІ .NET CORE

Киселевич В.В., студент, v.kyselevych@proton.me, ЖДУ імені Івана Франка
Усата О.Ю., к. п. н., доцент, o.y.usata@gmail.com, ЖДУ імені Івана Франка

Монолітна архітектура є традиційним простим рішенням розробки програмного забезпечення. Додаток містить в собі увесь необхідний функціонал в одній кодовій базі, призначений для одного сервера, наприклад, послуги для покупок, керування обліковими записами та навіть реклами. Розробка в цьому стилі, хоч і проста, але може мати ряд недоліків. Використання такого підходу може обмежувати швидкість та гнучкість розробки програми, оскільки зміни в одному модулі можуть впливати на всю систему, потребуючи більш детального планування та керування змінами. У разі помилки або відмови в одному модулі, це може призвести до зупинки всієї системи, оскільки всі компоненти програми залежать від одного сервера. Існує й ряд інших недоліків, які проявляються в деталях. Уникнути недоліків монолітної архітектури можна, застосувавши мікросервісну архітектуру, яка стає все більш популярною в розробці програмного забезпечення.

Мікросервісна архітектура є одним з найбільш ефективних способів розробки програмного забезпечення. Використання мікросервісної архітектури дозволяє розробникам побудувати систему, яка легко масштабується та підтримується, дозволяє розробляти нові функції швидше, зменшує час на виправлення помилок та забезпечує більшу гнучкість. Крім того, мікросервісна архітектура може допомогти зменшити ризик розробки програмного забезпечення, оскільки вона дозволяє розробникам легко відокремити та ізолювати компоненти системи. Це зменшує вплив помилок на інші частини системи та дозволяє швидко локалізувати та виправляти проблеми.

Мікросервісна архітектура – розділення кодової бази на кілька менших автономних програм [1]. Кожна програма діє як окрема маленька служба, звідки й назва мікросервіс. Кожний мікросервіс виконує певну, чітко визначену функцію, має власну кодову базу, сервер й іноді базу даних. Ці сервіси побудовані навколо бізнес-потреб (кожен відповідальний за конкретний процес), розробляються, розгортаються й масштабуються незалежно з використанням повністю автоматизованого середовища. Самі по собі сервіси можуть бути написані на різних мовах програмування і використовувати різні технології зберігання даних.

Переваги використання мікросервісної архітектури [2]:

- Масштабованість. Легко масштабувати окремі мікросервіси, що виконуються на окремих серверах. Це дозволяє забезпечити більшу продуктивність та ефективне використання ресурсів.

- Гнучкість. Можливість легко змінювати окремі мікросервіси без впливу на решту системи. Це дозволяє команді розробників швидко внести зміни та покращення, не турбуючись про відсутність сумісності з іншими частинами системи.
- Незалежність. Кожен мікросервіс може бути розроблений на різних мовах програмування, використовувати різні технології та зберігати дані в різних форматах. Це дозволяє розробникам використовувати найбільш підходящі технології для кожного мікросервісу та вільно комбінувати їх.
- Ресурси. Кожен мікросервіс має своє власне середовище виконання, що дозволяє більш ефективно використовувати ресурси сервера. Наприклад, якщо один мікросервіс потребує більшої кількості ресурсів, то можна запустити більше копій цього сервісу на окремих серверах, замість того, щоб запускати більший сервер для всього програмного забезпечення.
- Розширюваність. Мікросервісна архітектура дозволяє легко додавати нові функції до програмного забезпечення, без необхідності змінювати весь код.
- Менші ризики. Використання мікросервісної архітектури дозволяє зменшити ризик для всієї системи у разі виникнення проблем у одного з мікросервісів. Якщо один з мікросервісів перестане працювати, то це не вплине на інші мікросервіси.
- Легкість розгортання. Кожен мікросервіс може бути розгорнутий незалежно від решти системи. Це дозволяє зменшити час на розгортання та спростити процес розробки нових версій програмного забезпечення.

Мікросервісна архітектура має багато сильних сторони, проте, існують й деякі недоліки:

- Складність. Мікросервісна архітектура зазвичай призводить до більш складної системи, оскільки потрібно керувати більшою кількістю окремих сервісів та взаємодій між ними. Крім того, виникає проблема розподіленої системи, де кожен мікросервіс може використовувати різні технології та бібліотеки, що робить систему ще складнішою.
- Витрати. Розробка та підтримка мікросервісної архітектури може бути витратною у плані часу та коштів, оскільки потрібно забезпечувати кожен окремий мікросервіс, а також взаємодію між ними. До того ж, для підтримки такої системи потрібні спеціалісти, які розуміють архітектуру та можуть виконувати відповідні завдання.
- Низька продуктивність. Мікросервісна архітектура може мати низьку продуктивність через те, що мікросервіси мають взаємодіяти між собою через мережу, що може призводити до затримок у відповіді та зависання системи. Кожен окремий мікросервіс може мати свої власні проблеми з продуктивністю, що може впливати на всю систему.
- Координація. У мікросервісній архітектурі важливо забезпечити координацію між різними сервісами. Це може бути складно, особливо коли сервіси розробляються та підтримуються різними розробниками.

- Проблеми з безпекою. Мікросервісна архітектура може створювати проблеми з безпекою, особливо коли сервіси залежать від інших сторонніх сервісів. Це може збільшувати ризик кібератак та злому.

Складовими мікросервісної архітектури є:

- Мікросервіси. Основним компонентом архітектури мікросервісів є автономні сервіси. Вони можуть бути написані будь-якою мовою програмування та функціонувати окремо один від одного.
- Контейнери. Контейнери є важливою складовою мікросервісної архітектури, оскільки дозволяють упаковувати та розгортати мікросервіси зі всіма необхідними залежностями та середовищем, що дозволяє забезпечувати консистентність та незалежність мікросервісів.
- Service Discovery. У мікросервісній архітектурі, мікросервіси можуть бути дуже дрібними та їх може бути багато. Кожен з цих мікросервісів може запускатися на різних серверах або контейнерах. Service Discovery є важливою складовою мікросервісної архітектури, оскільки дозволяє забезпечити автоматичне відстеження та знаходження мікросервісів, що працюють у системі, та забезпечує ефективну взаємодію між ними.
- API Gateway. Даний компонент є важливою частиною зв’язку в складній розподіленій архітектурі мікросервісів, що розташований між клієнтськими програмами та внутрішніми мікросервісами. Бере на себе управління всіма зовнішніми запитами до мікросервісів та забезпечує їх безпеку й доступність. Керує доступом до мікросервісів, забезпечуючи автентифікацію та авторизацію користувачів. Захищає мікросервіси від різних видів атак, таких як DDoS або SQL-ін'єкції. Кешує результати запитів до мікросервісів, що дозволяє зменшити час відповіді та збільшити продуктивність системи.

В екосистемі C#, мікросервіси створюються з використанням .NET Core, кросплатформеного середовища для створення сучасних веб-додатків. .NET Core має багатий інструментарій і набір бібліотек для створення мікросервісів, включаючи вбудований веб-сервер, підтримку залежностей і бібліотеки для роботи з системами обміну повідомленнями. На додаток до баз даних для певного мікросервісу, архітектура може також містити в собі спільні бази даних, які використовуються кількома мікросервісами одночасно. Це можуть бути сховища даних для логування, кешування або зберігання спільних параметрів конфігурації тощо.

При побудові мікросервісної архітектури особливу увагу необхідно звернути на взаємодію між мікросервісами та контейнерезацію.

Одним з найпримітивніших способів організації взаємодії між мікросервісами є HTTP протокол. Даний протокол є досить простим, тому його дуже легко використовувати для забезпечення комунікації між мікросервісами. До того ж, є підтримка багатьох форматів даних, таких як: JSON, XML, HTML тощо, що дозволяє передавати дані між мікросервісами в різних форматах. Проте, для великої кількості запитів та об’ємів інформації може бути дещо повільним.

RabbitMq – покликаний стати ефективною альтернативою HTTP протоколу. RabbitMQ може забезпечити високу надійність, ефективність та масштабованість системи мікросервісів, а також забезпечити гнучкість та підтримку багатьох мов програмування й протоколів [3]. Схему роботи системи разом з технологією RabbitMq зображено на Рис. 2. Дана технологія має наступні особливості:

- дозволяє використовувати асинхронну комунікацію між мікросервісами, що дозволяє забезпечити більшу ефективність та швидкість обробки запитів;
- може зберігати повідомлення в режимі очікування, тому він може забезпечити високу надійність та масштабованість в системі мікросервісів;
- підтримує багато мов програмування;
- дозволяє забезпечити різні способи комунікації між мікросервісами;
- підтримує різноманітні протоколи, такі як: AMQP, MQTT, STOMP.

Для керування мікросервісами, зазвичай використовують технологію контейнеризації. Це дозволяє упаковувати кожну службу з її залежностями в легкий контейнер, який можна легко розгортати та масштабувати. Однією з найпопулярніших технологій контейнерезації є Docker [4].

Особливості використання Docker:

- контейнери забезпечують ізольоване середовище для кожного мікросервісу, що забезпечує більшу надійність та безпеку у випадку виникнення помилок або атак на систему;
- дозволяє розгортати та масштабувати мікросервіси швидко та ефективно. Це може бути особливо корисним у випадку, коли потрібно додатково розгорнути екземпляри сервісів для забезпечення більшого обсягу трафіку;
- дозволяє розробникам швидко розгортати та випробовувати різні версії мікросервісів безпосередньо на своєму локальному комп'ютері. Це дозволяє швидше та ефективніше розробляти та тестувати нові функції;
- надає можливість запускати більше мікросервісів на одному фізичному сервері, що може знизити витрати на інфраструктуру;
- забезпечує зручний інтерфейс для керування мікросервісами, включаючи запуск, зупинку та моніторинг.

Інструменти оркестровки контейнерів, такі як Kubernetes або Docker Swarm, можна використовувати для керування розгортанням і масштабуванням контейнерів на кількох хостах або кластерах.

Висновок. Таким чином, мікросервісна архітектура дозволяє розбити монолітні програми на менші компоненти, якими легше керувати. Кінцевою метою переходу на мікросервісну архітектуру є реалізація розподіленої, слабозв’язаної та незалежної структури. Провідні технологічні компанії, такі як: Amazon, Netflix і Uber, використовують мікросервіси для ефективної роботи, уникаючи серйозних операційних проблем. Хоча мікросервісна архітектура має численні переваги використання, перехід може бути складним. Застосування мікросервісної архітектури в маленьких системах, може призвести тільки до

складнощів, на відміну від великих систем, де використання даного підходу буде хорошою інвестицією, яка в подальшому полегшить розробку.

Список використаних джерел

1. Microservice Architecture: Aligning Principles, Practices, and Culture / Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, Mike Amundsen. O’Reilly Media. 2016. С. 4-11.
2. Architectural Patterns for Microservices: A Systematic Mapping Study / D Taibi, V Lenarduzzi and Claus Pahl / Proceedings of the 8th International Conference on Cloud Computing and Services Science; Funchal, Madeira, Portugal, 19-21 March 2018.
3. Документація RabbitMq. URL: https://www.rabbitmq.com (дата звернення 02.12.2023)
4. Документація Docker. URL: https://www.docker.com (дата звернення 02.12.2023)

УДК 004

РОЗРОБКА МЕТОДУ ЗМЕНШЕННЯ РОЗМІРНОСТІ UMAP НА ТЕХНОЛОГІЇ WEBGPU

Ковальов Д.О., студент, k18_kovalov.do@server.odessa.ua, МАУП
Шибаєва Н.О., к. т. н., доцент, shibaeva@server.odessa.ua, коледж «Сервер»

На сьогоднішній день, в сфері досліджень та інновацій, велике значення надається аналізу великих обсягів даних для розуміння та оптимізації різноманітних явищ. Цей процес застосовується до вивчення різних аспектів, таких як соціальні тенденції, екологічні виклики, технологічні інновації та інші сфери.

Одним із таких процесів є створення нових лікувальних засобів. Даний процес спирається на аналіз великих об'ємів даних, зокрема молекул, пептидів, тощо. Невід'ємною частиною цього процесу є використання методів зменшення розмірності та кластеризації багатовимірних даних для знаходження груп молекул з подібними функціями.

Обробка відповідних наборів даних, що зазвичай містять не менше 100,000 елементів, потребує значних обчислювальних ресурсів та витрат пам'яті. Так, при використанні наявних програмних засобів для аналізу типового набору необхідні зберігання та обробка допоміжної матриці дистанцій, яка складається мінімум з 10 млрд. елементів. З застосуванням наявних можливостей обробка