Технічні науки

УДК 004.852

Зеленін Віктор Анатолійович
бакалавр комп'ютерних наук
Національного технічного університету України
«Київський політехнічний інститут імені Ігоря Сікорського»
Зеленин Виктор Анатольевич
бакалавр компьютерных наук
Национального технического университета Украины
«Киевский политехнический институт имени Игоря Сикорского»
Zelenin Victor
bachelor of computer science of
The National Technical University of Ukraine
«Kyiv Polytechnic Institute»

ОСНОВНІ ПРИНЦИПИ ПОБУДОВИ МІКРОСЕРВІСНИХ СИСТЕМ
ОСНОВНЫЕ ПРИНЦИПЫ ПОСТРОЕНИЯ МИКРОСЕРВИСНЫХ СИСТЕМ
BASIC APPROACHES OF BUILDING MICROSERVICE SYSTEMS

Анотація. Робота присвячена дослідженню основних підходів до побудови мікросервісної архітектури, а також дослідження існуючих концепцій до створення мікросервісних систем, ознайомлення з наявними програмними платформами та інструментами.

Ключові слова: мікросервісна архітектура, сервіс-орієнтована архітектура, SOAP, API, REST.

Аннотация. Работа посвящена исследованию основных подходов к построению микросервисной архитектуры, а также исследование имеющихся программных платформ и инструментов.

Ключевые слова: микросервисная архитектура, сервис-ориентированая архитектура, SOAP, API, REST.

Summary. Thesis is devoted to the researching main approaches of building microservice architecture, investigating existing principles for creating microservice systems and acquanting with the available programming platforms and tools.

Key words: microservice architecture, service oriented architecture, SOAP, API, REST.

Проблема проектування та створення якісного програмного забезпечення є надзвичайно важливою у сучасному інформаційному світі. З розвитком IT- індустрії було знайдено багато різних підходів та концепцій до побудови складних програмних систем.

Довгий час провідне місце займала так звана "монолітна архітектура". При даному підході вся система являє собою моноліт, який фізично розташовується на єдиній машині, запускається в одному процесі та виконує всі бізнес-операції системи.

Монолітний додаток піддається лише горизонтальному масштабуванню шляхом запуску декількох окремих серверів із кожним окремим монолітом. Але з плином часу знаходилися інші ідеї та підходи, саме таким стала сервіс-орієнтована архітектура (SOA), на відміну від монолітної системи, при SOA вся програма являє собою розподілену систему, яка обмінюється повідомленнями за певним протоколом. Вся система складається з набору незалежних сервісів, які фокусуються на власній задачі. SOA націлена на боротьбу з великими монолітними системами. Сама по собі ідея SOA чудова, але питання як правильно та якісно організовувати сервіс-орієнтовану архітектуру залишається відкритим. Основні вузькі місця відносяться до протоколів обміну даними, таких як SOAP, а також неправильні місця розділу системи.

Пізніше було запропоновано новий підхід до організації SOA, так звана мікросервісна архітектура (MSA). Мікросервісну архітектуру можна вважати

підмножиною SOA, але все ж таки MSA відрізняється від класичного SOA, основна відмінність — це невелика кількість кодової бази на кожен сервіс, в той час як в SOA не важливий об’єм кодової бази. Також важливим місцем для MSA — це те, що кожен сервіс має мати власний обмежений контекст для цієї предметної області.

Обмежень на кількість існуюючих сервісів немає, але кожен сервіс має працювати лише над одною бізнес-задачею. Для обміну інформацією мікросервіси використовують стандартизовані протоколи передачі даних (наприклад, HTTP), як правило кожен сервіс має своє API для спілкування з іншими мікросервісами. Всі сервіси можуть бути написані на абсолютно різних мовах програмування та використовуючи будь-які фреймворки, також має місце децентралізоване збереження даних, тобто кожен сервіс має свою власну базу даних [1].

В таблиці 1 наведено ключові переваги монолітної та мікросервісної архітектур.

Таблиця 1 – Основні переваги трьохрівневої архітектури та SOA

Монолітна архітектура	**Мікросервісна архітектура**
Простота	Часткове розгортання
Узгодженість	Відмовостійкість
Міжмодульний рефакторинг	Відсутність стану
	Гетерогенність

В цілому, мікросервісну архітектуру вважають підмножиною SOA, тому що вона, аналогічно SOA, орієнтована на сервіси та розподілена, використовує унифіковані протоколи обміну даними. MSA слід розглядати як підхід до реалізації SOA.

Ідеологія мікросервісів закликає використовувати розумні приймачі та прості канали передачі, замість складних протоколів, типу WS-* або BPEL, слід використовувати REST протоколи [2]. Найпопулярніші протоколи – це

HTTP запити та неперевантажений меседжинг. До легких протоколів меседжингу відносяться: AMQP, STOMP, RabbitMQ, Apache Kafka, NSQ.

Важливим аспектом будь-якої інформаційної системи є організація та управління даними. При мікросервісному підході практикується децентралізоване управління даними, приклад якого представлено на рисунку 1. Для стандартного монолітного додатку існує лише одна база даних, яка обслуговує безліч різних компонентів бізнес-логіки системи.

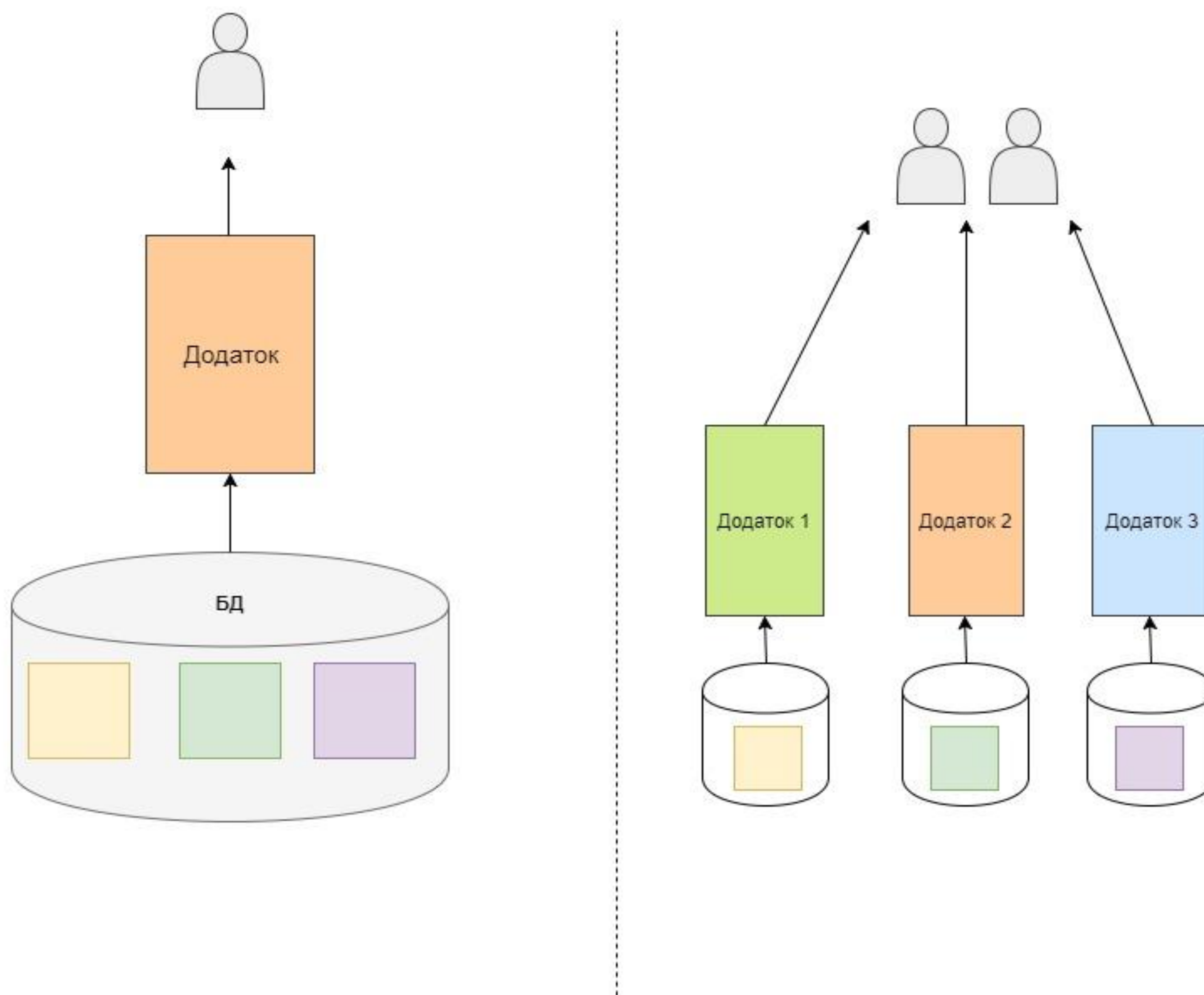

Рисунок 1. Порівняння монолітного та мікросервісного підходів до управління даними

Найважливішим технічним аспектом під час проектування та реалізації мікросервісів є правильна організації взаємодії окремих компонентів. Найпростішим підходом до інтеграції сервісів є агрегування, тобто є один

ключовий сервіс, який викликає безліч інших сервісів, схема даного шаблону наведена на рисунку 2.

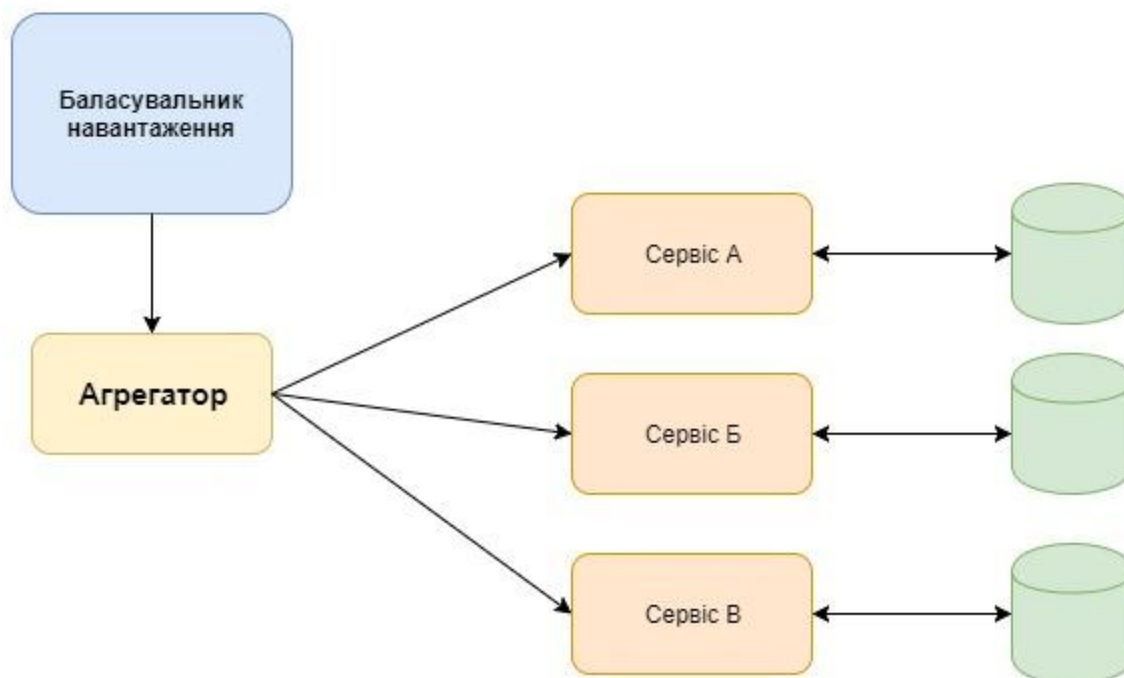

Рисунок 2. Схема шаблону «Агрегатор»

Даний патерн дотримується принципу DRY. Якщо існує безліч сервісів, які звертаються до сервісів А, Б, і В, то рекомендується абстрагувати цю логіку в мікросервіс та агрегувати її у вигляді окремого сервісу.

При всій поширеності та зрозумілості вище описаного підходу, у нього є важливе обмеження, а саме: він синхронний і, отже, блокуючий. Забезпечити асинхронність можна, але це робиться по-своєму в кожному додатку. Тому в деяких мікросервісних архітектурах можуть використовуватися черги повідомлень, а не модель REST запит / відповідь. На рисунку 3 зображено як сервіс А може синхронно викликати сервіс В, який потім буде асинхронно зв'язуватися з сервісами Б, Д за допомогою черги повідомлень.

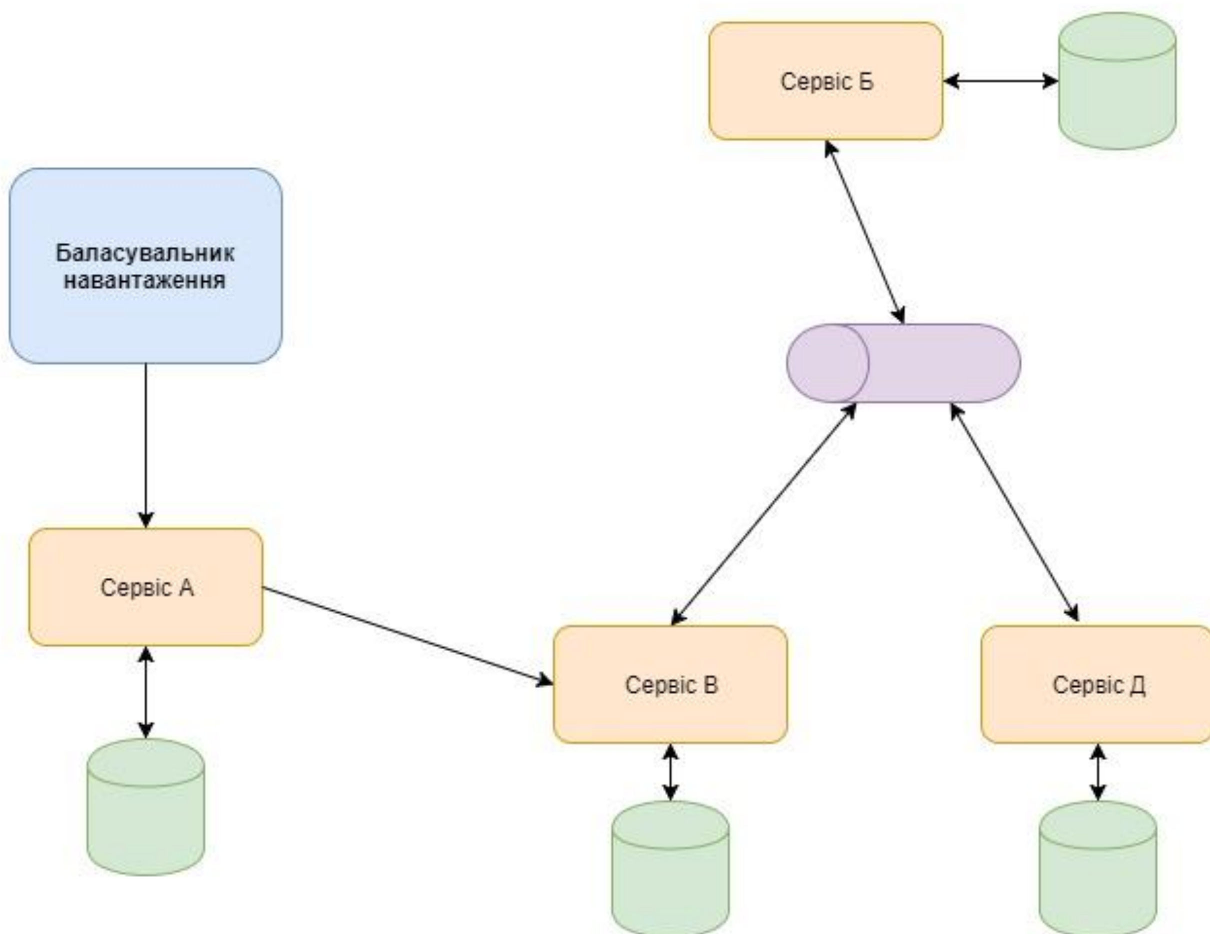

Рисунок 3. Схема шаблону з чергою повідомлень

Комбінація моделі синхронних та асинхронних викликів також можуть використовуватися для досягнення поставлених задач.

Для успішної реалізації взаємодії мікросервісів та всієї внутрішньої роботи необхідно використовувати додаткові інструменти, які будуть описані нижче.

Мікросервісна система може складатися з десятків або, можливо, навіть сотень сервісів, тому клієнту досить тяжко взаємодіяти з всіма мікросервісами індивідуально. Більше того, навіть неможливо знайти всі мікросервіси без додаткових інструментів. У такій ситуації було прийнято використовувати шаблон API Gateway. Даний патерн виступає в якості єдиної точки входу для набору пов’язаних мікросервісів або навіть всієї системи [3].

Переваги використання спільної точки входу [4]:

- Єдиний адрес набагато зручніший сотні індивідуальних адресів API
- Зручно реалізовувати обмеження на кількість запитів в єдиному місці
- Вся система стає гнучкішою – можна змінювати внутрішню структуру
- Можна кешувати відповіді
- Можна поєднувати відповіді від різних сервісів.

У мікросервісному середовищі, екземпляри регулярно додаються або видаляються при масштабуванні. Оскільки сервери та порти часто призначаються автоматично, клієнти, а також інші мікросервіси повинні бути в змозі знайти та ідентифікувати їх, щоб взаємодіяти. Це завдання вирішується за допомоги концепції виявлення сервісів, в якому ключовим є компонент, який називають реєстром сервісів, що відстежує всі доступні мікросервіси в системі. Кожен сервіс реєструється при запуску, використовуючи клієнт на самому сервісі, який здійснює зв'язок з реєстром, або через сторонній додаток, який контролює екземпляри сервісів в середовищі та зберігає свій статус в реєстрі.

Мікросервіси часто співпрацюють при опрацюванні запитів. Коли один сервіс викликає синхронно інший, завжди є можливість того, що сервіс буде недоступним або матиме великий час відгуку. Це може привести до виснаження ресурсів, які можуть зробити викликаючий сервіс не здатним опрацьовувати інші запити. Відмова одного сервісу може каскадним чином зупинити роботу всієї системи [5].

Для забезпечення надійності роботи системи використовують шаблон автоматичний вимикач. За даним патерном, необхідно обернути виклик об'єктом автоматичного вимикача, який буде стежити за виключними ситуаціями в системі. Як тільки несправність досягає деякого порогу, то вимикач спрацьовує та всі подальші виклики до вимикача повертаються з помилкою, а перший виклик не виконується взагалі.

У даній роботі було розглянуто найважливіші аспекти та головні шаблони реалізації мікросервісних додатків, яких рекомендовано дотримуватися для побудови ефективних програм. Також було здійснено огляд основних інструментів для забезпечення правильного функціонування всієї інфраструктури мікросервісів.

Використавши всі вище описані концепції, розроблений додаток повинен правильно та ефективно працювати, а також володіти необхідною відмовостійкістю.

Література

1. Ньюмен С. Создание микросервисов / Ньюмен С. – СПб.: Питер, 2016 – 304 с.
2. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions / Gregor Hohpe, Bobby Woolf – Addison-Wesley, 2004 – 736 p.
3. Using an API Gateway. – Режим доступу: https://nginx.com/blog/building-microservices-using-an-api-gateway/ – Дата доступу: 27.07.2017
4. Service Discovery. – Режим доступу: https://nginx.com/blog/service-discovery-in-a-microservices-architecture/ – Дата доступу: 27.07.2017
5. Chris Richardson. From Design to Deployment / Chris Richardson, Floyd Smith, 2016. – 74 p.