УДК 004.273 DOI: https://doi.org/10.30837/IYF.IIS.2024.673

ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПРОЦЕССІВ ВЗАЄМОДІЇ СЕРВІСІВ В МІКРО-СЕРВІСНІЙ АРХІТЕКТУРІ

Белименко В. С.
Науковий керівник – к.т.н., проф. Іванов В. Г.
Харківський національний університет радіоелектроніки, каф. СТ
м. Харків, Україна
e-mail: viacheslav.belymenko@nure.ua

In the digital era, microservices architecture emerges as a key for developing scalable, flexible, and reliable systems. It allows for the independent updating and deployment of system components, enhancing development efficiency and system resilience. A primary challenge in microservices architecture is service interaction, encompassing effective communication, dependency management, data consistency, and transaction pattern implementation. Overcoming these challenges requires understanding service interaction nuances, including network communication and resilience strategies.

У сучасному світі цифрових технологій, мікросервісна архітектура набуває все більшого розповсюдження як ключовий елемент у розробці та підтримці масштабованих, гнучких і надійних систем. Втім, розподіленість системи вносить додаткову складність у взаємодію між сервісами, що потребує детального дослідження та аналізу.

Однією з ключових проблем в мікросервісній архітектурі є взаємодія між сервісами. Це включає в себе проблематику забезпечення ефективної комунікації, управління залежностями, забезпечення консистентності даних та реалізацію шаблонів транзакцій. Вирішення цих проблем вимагає глибокого розуміння різних аспектів взаємодії сервісів, включаючи мережеву взаємодію, шаблони проектування та стратегії відмовостійкості [1].

У мікросервісній архітектурі взаємодія між сервісами може бути класифікована на більш загальні типи, які охоплюють різні способи комунікації та інтеграції сервісів. Основними типами взаємоії являються синхронна та асинхронна комунікація.

Синхронна взаємодія між сервісами відбувається, коли один сервіс викликає інший і чекає на негайну відповідь перед продовженням своєї операції. Цей підхід є блокуючим, тобто викликаючий сервіс не може виконувати інші завдання, поки не отримає відповіді. Синхронна взаємодія часто використовується для операцій, які вимагають негайного результату або підтвердження. До таких типів відносяться:

– REST популярний архітектурний стиль для розробки веб-сервісів. Він використовує HTTP протокол для комунікації, де сервіси взаємодіють через стандартні HTTP методи, такі як GET, POST, PUT, DELETE. REST вважається легковаговим і простим у використанні;

– SOAP один із головних конкурентів REST. Це протокол обміну повідомленнями, який дозволяє програмам спілкуватися між собою через HTTP або інші протоколи. SOAP повідомлення є строго структурованими і зазвичай використовують XML для форматування даних;

– gRPC являється високопродуктивним фреймворком відкритого коду, розроблений Google, для виклику процедур між сервісами;

– GraphQL є мовоє запитів для API, яка дозволяє клієнтам точно вказувати, які дані їм потрібні. Вона дозволяє агрегувати запити до різних ресурсів в один запит, що може зменшити кількість запитів і поліпшити продуктивність. GraphQL використовується для синхронної взаємодії і може служити потужним інструментом для розробки флексибільних API.

Асинхронна взаємодія між сервісами в мікросервісній архітектурі відбувається, коли один сервіс відправляє запит або повідомлення іншому сервісу без очікування на негайну відповідь. Це дозволяє викликаючому сервісу продовжувати свою роботу без блокування, поки не буде оброблено повідомлення. Асинхронна комунікація є ключовою для підвищення масштабованості та відмовостійкості системи, оскільки вона знижує залежність між сервісами. Ось кілька типів асинхронної взаємодії:

– Message Queues дозволяють сервісам відправляти та отримувати повідомлення через посередника, який зберігає повідомлення до їх обробки. Це забезпечує відмовостійкість, оскільки повідомлення не втрачаються при відмові сервісу;

– Topics/Subscriptions дозволяють сервісам публікувати повідомлення в теми, а інші сервіси підписуються на ці теми для отримання повідомлень. Це сприяє створенню розподілених публікаційно-підписних систем, які можуть ефективно масштабуватися.

– Event-Driven Architecture – в цьому підході сервіси генерують події, які інші сервіси можуть споживати. Це створює зв'язану систему, де сервіси можуть реагувати на події в системі без прямого зв'язку між виробником та споживачем.

Список використаних джерел:

1. Стьопін В. І., Горбатенко Б. В., Іванов В. Г. Впровадження мікросерввісної архітектури у високонавантажені сучасні web-системи// Міжнародна науково-практична конференція «Інформаційні технології в соціокультуній сфері,освіті та економіці» Частина 1. Київ, 2019 . С. 88-90.