### ЛЕКЦІЯ 11. МІКРОСЕРВІСНА АРХІТЕКТУРА

**1. Вступ до мікросервісної архітектури**

Мікросервісна архітектура — це підхід до розробки програмного забезпечення, при якому додаток складається з набору невеликих, незалежних сервісів, що забезпечують окремі функції. Цей підхід дозволяє кожному мікросервісу бути незалежним компонентом, який можна розробляти, тестувати, розгортати та масштабувати окремо від інших. Мікросервіси часто взаємодіють один з одним через легковагові протоколи, такі як HTTP, WebSocket або API на основі протоколу REST.

Традиційні монолітні добавки будувалися як єдина система, що ускладнювало їх розвиток і підтримку в умовах складності й динамічних змін. У такому разі мікросервіси не дозволяють зробити добавки більш гнучкими, адаптивними та стійкими до збоїв, оскільки кожен сервіс є самостійним модулем.

**2. Основи мікросервісів**

**2.1. Що таке мікросервіс?**

Мікросервіс — це незалежний компонент системи, який виконує певну логіку або функціонал і зв'язується з іншими компонентами через добре визначені інтерфейси. Кожен мікросервіс має свої дані, логіку і навіть базу даних (якщо це необхідно), що робить його повністю автономним. Такий підхід значно спрощує керування окремими компонентами та дозволяє легко підтримувати, оновлювати або замінювати один сервіс, не торкаючись інших.

**2.2. Мікросервіси vs Моноліт**

Монолітна структура конструкції, що всі компоненти системи пов'язані і розгортаються як єдине ціле. Це створює низку проблем у контексті масштабування, оновлення та розвитку продукту. Наприклад, навіть якщо оновлення великих незначних функцій вимагає перезавантаження в усі системи, є великий ризик для проектів.

Мікросервісна архітектура, у своєму разі, дозволяє:

* Розробити окремі частини незалежної програми.
* Використовувати різні технології для кожного мікросервісу.
* Підтримувати гнучко масштабування кожного окремого компонента.
* Мати чітко визначені межі відповідальності кожного сервісу.

**2.3. Принципи мікросервісної архітектури**

* **Автономія послуг** : кожен сервіс має бути незалежним і мати можливість розроблятися і розгортатися автономно.
* **Декомпозиція за бізнес-функціями** : сервіси повинні бути орієнтовані на конкретні бізнес-процеси.
* **Дистрибутивність** : сервіси повинні функціонувати незалежно від того, на яких платформах або інфраструктурі вони розгорнуті.
* **Можливість масштабування** : кожен сервіс може бути масштабований незалежно від інших.
* **Малий обсяг** : сервіси мають бути компактними і відрізняються однією чітко визначеною функцією.

**3. Переваги мікросервісної архітектури**

Мікросервісний підхід має низку переваг, які роблять його популярним вибором для сучасних розподілених систем.

**3.1. Гнучкість у розробці та розгортанні**

Однією з головних переваг мікросервісів є можливість незалежного розвитку та розгортання. Це означає, що кожен сервіс може бути оновлений або вдосконалений без необхідності зупиняти всю систему. Наприклад, якщо один мікросервіс потребує вдосконалення або виправлення помилок, це можна зробити без негативного впливу на роботу інших частин системи.

**3.2. Масштабованість**

Мікросервіси не можуть масштабувати окремі компоненти системи залежно від навантаження. Наприклад, якщо сервіс автентифікації користувачів має велике завантаження, його можна легко масштабувати горизонтально, не зачіпаючи інші компоненти. Це забезпечує гнучкість і економічну ефективність в управлінських ресурсах.

**3.3. Підтримка різноманітних технологій**

Оскільки кожен мікросервіс є незалежною одиницею, команди розробників можуть вибирати для кожного сервісу оптимальні технології та інструменти. Наприклад, один сервіс може бути написаний на Python, а інший — на Java, якщо ці мови найкраще підходять для конкретних завдань.

**3.4. Стійкість до збоїв**

Завдяки розподіленості систем з мікросервісами, збої в одному сервісі не призводять до падіння в усі системи. Наприклад, якщо служба обробки платежів виходить з ладу, інші служби, такі як реєстрація користувачів або перегляд контенту, продовжують працювати безперебійно.

**4. Недоліки мікросервісної архітектури**

Хоча мікросервісна архітектура має багато переваг, вона також має і ваші недоліки.

**4.1. Складність управління**

Розробка та підтримка мікросервісів вимагає від розробників розуміння розширених систем. Координація між сервісами, управління версіями API та налаштування серверів може стати складним завданням.

**4.2. Ускладнене тестування**

Тестування систем з мікросервісами є складним завданням із монолітними системами. Потрібно тестувати не тільки окремі сервіси, але й їх взаємодію між собою.

**4.3. Витрати на інфраструктуру**

Мікросервісна архітектура часто потребує більше ресурсів для хостингу та обслуговування через підтримку кількох незалежних компонентів. Це може збільшити витрати на інфраструктуру, особливо якщо потрібно розгортати та підтримувати десятки чи навіть усі мікросервіси.

**5. Оркестрація мікросервісів**

Оркестрація мікросервісів підтримує процес управління життєвим циклом, розгортанням та комунікацією між мікросервісами. Одним із найпопулярніших інструментів для оркестрування є **Kubernetes**. Kubernetes дозволяє автоматизувати процес розгортання, управління та масштабування контейнерів, в яких працюють мікросервіси.

**5.1. Основні функції оркестрування:**

* **Управління розгортанням** : автоматизація процесу розгортання нових версій мікросервісів.
* **Моніторинг** : відстеження стану послуг і забезпечення їх безперебійної роботи.
* **Масштабування** : автоматичне збільшення або зменшення кількості запущених екземплярів служб залежно від навантаження.
* **Забезпечення взаємодії між сервісами** : автоматизація мережевої взаємодії між мікросервісами.

**6. Комунікація між мікросервісами**

У системах на базі мікросервісів основним методом взаємодії є **API** (Application Programming Interface). Кожен мікросервіс надає свій інтерфейс для взаємодії з іншими сервісами, що дозволяє їм обмінюватися даними.

**6.1. Синхронна комунікація**

Вид комунікації здійснюється в режимі реального часу. Наприклад, сервіс замовлень може надіслати запит на сервіс платежів і отримати відповідь про успішну обробку. Основний недолік цього підходу — підтримка у відповідності або можливості того, що один із сервісів може стати недоступним, що порушить роботу в усій системі.

**6.2. Асинхронна комунікація**

У цьому випадку сервіс обміну повідомленнями здійснюється через черги повідомлень або брокерів (наприклад, RabbitMQ, Apache Kafka). Це дозволяє виконувати завдання незалежно один від одного. Наприклад, сервіс замовлень може відправити запит на обробку платежу і продовжити роботу без очікування відповіді від сервісу платежів.

**7. Приклади використання мікросервісної архітектури**

**7.1. Netflix**

Netflix є однією з найвідоміших прикладних компаній, яка використовує мікросервісну структуру для підтримки своєї платформи потокового мовлення. Завдяки мікросервісам Netflix може окремо масштабувати сервіси для потокового відео, рекомендацій, авторизації користувачів тощо, забезпечуючи високу надійність і продуктивність систем.

**7.2. Amazon**

Amazon також активно використовує мікросервіси для свого електронного магазину. Кожен компонент системи, від обробки замовлений до каталогу управління, є окремим мікросервісом, що дозволяє легко керувати масштабуванням та оновленням функціональності.

Мікросервісна архітектура стала потужним інструментом для побудови складних, масштабованих та гнучких систем. Вона забезпечує високу гнучкість у розробці, розгортанні та масштабуванні, хоча вимагає від команди розробників знань про розподілені системи та оркестрування. Зрозуміти основні принципи мікросервісів, їх переваги та недоліки, а також інструменти для оркестрації є критично ефективними для успішної реалізації сучасних додатків.