УДК 621.39

ПОРІВНЯЛЬНЕ ДОСЛІДЖЕННЯ АРХІТЕКТУРНИХ РІШЕНЬ ДЛЯ ПОБУДОВИ МАСШТАБОВАНИХ ПРОГРАМНИХ ПРОДУКТІВ

Дуля О.О., Міночкін Д.А.
Інститут телекомунікаційних систем КПІ ім. Ігоря Сікорського, Україна
E-mail: sasa97973@gmail.com

THE COMPARATIVE STUDY OF ARCHITECTURAL SOLUTIONS FOR MAKING SCALABLE SOFTWARE PRODUCTS

This work about researching architectures serverless and microservices. It shows the difference between these architectures. The report reveals the biggest advantages and disadvantages of serverless and microservices.

Сервіс-орієнтована архітектура (*service-oriented architecture,* SOA) придумана в кінці 1980-х. Вона бере свій початок в ідеях, викладених в CORBA (*Common Object Request Broker Architecture* — загальна архітектура брокера об'єктних запитів), DCOM (*Distributed Component Object Model* - розподілена модель компонентних об'єктів), DCE (*Data Communication Equipment* - Апаратура передачі даних) і інших документах [3]. Про SOA написано багато, є кілька її реалізацій. Але, по суті, SOA можна звести до кількох ідей, причому архітектура не диктує способи їх реалізації:

- Сполучуваність додатків, орієнтованих на користувачів.
- Багаторазове використання бізнес-сервісів.
- Незалежність від набору технологій.
- Автономність (незалежні еволюція, масштабованість і швидке розгортання).

SOA - це набір архітектурних принципів, що не залежать від технологій і продуктів, зовсім як поліморфізм або інкапсуляція.

В основі мікросервісної архітектури лежать концепції SOA. Призначення у неї: створити єдиний загальний корпоративний додаток з декількох спеціалізованих додатків бізнес-доменів [4].

Мікросервісна архітектура створювалася в контексті швидко і постійно мінливих бізнесів, які (в основному) з нуля створюють власні хмарні додатки. У випадку з мікросервісами ми повністю контролюємо додатки (при цьому в системі можуть використовуватися і сторонні веб-сервіси).

Характер побудови / проектування мікросервісів не вимагає глибокої інтеграції. Мікросервіси повинні відповідати бізнес-концепції, обмеженому контексту. Вони повинні зберігати свій стан, бути незалежними від інших мікросервісів, і тому вони менше потребують інтеграції. Тобто низька

взаємозалежність і висока зв'язність привели до побічного ефекту - зменшення потреби в інтеграції.

Тобто, мікросервіси - це маленькі автономні сервіси, що працюють разом і спроектовані навколо бізнес-домену.

Оскільки не обов'язково володіти серверами як фізичними машинами, в більшості випадків, розробник не повинен думати про сервери і масштабування, він має можливість зосередитися лише на бізнес-логіці.

Serverless дає змогу використовувати кожен з «мікросервісів» вашої програми, будь то платежі, управління користувачами, новинний бюлетень, як повністю незалежну функцію, яка може «впасти», не пошкодивши додаток цілком [1]. Кожна з цих служб може масштабуватися незалежно. Тобто якщо ніхто не використовує генерацію PDF-файлів, ця служба не доступна, але якщо багато користувачів вирішили скористатися цим сервісом, вона миттєво масштабується для його підтримки.

Також, ще однією перевагою є потрібність платити тільки за ті послуги, які використовуються, і тільки за час, коли вони дійсно використовувалися. Це і є Serverless [2].

Зазначу, що для Serverless потрібно трохи інша архітектура наших додатків. Вони повинні мати подієву структуру і не використовувати інформацію про стан (stateless).

Serverless і Мікросервіси мають спільні характеристики, але вони сильно відрізняються тим, як вони забезпечують хмарні обчислення.

Ціноутворення - це найперша відмінна риса. Serverless - це pay-as-you-go (плата, якщо використовується), що робить його привабливим варінтом для додатків, що мають незначну кількість запитів, або для стартап-організацій. Це також зменшує експлуатаційні витрати за рахунок того, що інфраструктура та віртуальні машини оброблюються постачальниками послуг, а не безпосередньо на власному обладнанні, IaaS (інфраструктура, як послуга) чи PaaS (платформа, як послуга).

Масштабування - це також перевага архітектури serverless. За своєю природою serverless може краще справитись з піковими навантаженнями і може автоматично, швидко керуватися. Єдиний недолік - це максимальна кількість запитів, які можуть бути оброблені, що робить її непридатною для високонавантажених систем.

Serverless ще більш детальна за мікросервіси і надає значно більшу степінь функціональності. Але зворотна сторона цього - це робить її більш складною. Serverless має також інші недоліки, такі як обмеження кількості запитів, обмежений час операції та підтримує меншу кількість мов програмування.

Його компонентами є часто конкретними постачальникам хмарних послуг, які може бути важко змінити. Функції мають бути stateless (без запам’ятовування стану), а керування станом відбуватися виключно, за границями функції - це означає, що більше ніякої кеш-пам’яті.

Окрім цього, функції в serverless можуть використовуватися, як планові задачі, чи обробники подій, а не лише в якості послуг.

Хоча мікросервіси все ще забезпечують ґрунтовний підхід до сервіс-орієнтованої архітектури, serverless розвивається в напрямку архітектури заснованої на подіях (event-based) і очевидно має переваги з точки зору часу до виходу на ринок, гнучкого ціноутворення та знижених експлуатаційних витрат. Малоймовірно, що serverless зараз підійде до кожної системи. Зараз, найкращим рішенням буде поєднання цих двох архітектурних рішень, для того, щоб донести і використати переваги хмарних технологій для себе і ваших клієнтів.

Таким чином, в роботі проведено порівняльний аналіз архітектурних рішень (serverless та мікросервісів), з чого випливає наступне:

1. Головна різниця serverless та мікросервісів в тому, як вони забезпечують хмарні обчислення.

2. Масштабованість та ціноутверення - головні переваги Serverless.

3. Мікросервіси не мають обмеженої кількості запитів та обмеженого часу операції, на відміну від Serverless.

4. Всі функції в serverless мають бути stateless (без запам’ятовування стану).

5. Мікросервіси забезпечують грунтовний підхід до сервіс-орієнтованої архітектури, а serverless розвивається в напрямку архітектури заснованої на подіях (event-based).

6. Найкращим варіантом, на сьогодні, є метод комбінування цих архітектурних рішень, для отримання максимальної вигоди.

Література

1. Roberts, Mike (25 July 2016). "Serverless Architectures". *MartinFowler.com*. Retrieved 30 July 2016.
2. "Creating full-stack Serverless applications on the AWS platform". *Serverless-Stack.com*. Retrieved 5 April 2017.
3. S. Newman, Building Microservices – Designing Fine-Grained Systems, O’Reilly, 2015.
4. E. Wolff, Microservices: Flexible Software Architecture, Addison-Wesley, 2016.