Інформаційні технології

УДК 004.852

Кутовий Олександр Олександрович
бакалавр комп'ютерних наук
Національного технічного університету України
«Київський політехнічний інститут»
Кутовой Александр Александрович
бакалавр компьютерных наук
Национального технического университета Украины
«Киевский политехнический институт»
Kutoviy O.
Bachelor of computer science
The National Technical University of Ukraine
«Kyiv Polytechnic Institute»

ЗАСТОСУВАННЯ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ ПРИ РОЗРОБЦІ СЕРВЕРНОЇ ЧАСТИНИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

ПРИМЕНЕНИЕ МИКРОСЕРВИСНОЙ АРХИТЕТУРЫ В РАЗРАБОТКЕ СЕРВЕРНОЙ ЧАСТИ ИНФОРМАЦИОННОЙ СИСТЕМЫ

DEVELOPING INFORMATION SYSTEM SERVER-SIDE WITH MICROSERVICES ARCHITECTURE

Анотація: Реалізовано серверну частину інформаційної системи на базі мікросервісної архітектури та з застосуванням контейнеризації та неперервної інтеграції.

Ключові слова: Сервер, контейнер, Doker, неперервна інтеграція, мікросервіс, agile.

Аннотация: Реализована серверная часть информационной системы на базе микросервисной архитектуры с применением контейнеризации и непрерывной интеграции.

Ключевые слова: Сервер, контейнер, Doker, непрерывная интеграция, микросервис, agile.

Abstract: Developed information system server-side based on microservices architecture with usage of containerization via Doker and continuous integration.

Keywords: Server, container, Doker, continuous integration, microserervice, agile.

На даний момент в індустрії наявні сильні тенденції до побудови систем шляхом з'єднання разом різних компонент, багато де - так само, як це відбувається в реальному світі. За останні пару десятків років ми бачили велике зростання набору бібліотек, використовуваних в більшості мов програмування.

Говорячи про компоненти, ми стикаємося з труднощами визначення того, що саме є компонентом. Запропоноване таке визначення: компонент - це одиниця програмного забезпечення, яка може бути незалежно замінена або оновлена.

Архітектура мікросервісів використовує бібліотеки, але їх основний спосіб розбиття додатку - шляхом ділення його на сервіси. Ми визначаємо бібліотеки як компоненти, які підключаються до програми і викликаються нею в тому ж процесі, в той час як сервіси - це компоненти, що виконуються в окремому процесі і комунікують між собою через веб-запити або remote procedure call (RPS).

Головна причина використання сервісів замість бібліотек - це незалежне розгортання. При розробці додатку, що складається з декількох бібліотек, які працюють в одному процесі, будь-яка зміна в цих бібліотеках призводить до перерозгортки всієї програми. Але якщо додаток розбитий на кілька сервісів, то зміни, що зачіпають будь-які з них, будуть вимагати

перерозгортання тільки зміненого сервісу. Звичайно, якісь зміни будуть зачіпати інтерфейси, що, в свою чергу, потребують певної координації між різними сервісами, але мета хорошої архітектури мікросервісів - мінімізувати необхідність в такій координації шляхом установки правильних кордонів між мікросервісами, а також механізму еволюції контрактів сервісів [8].

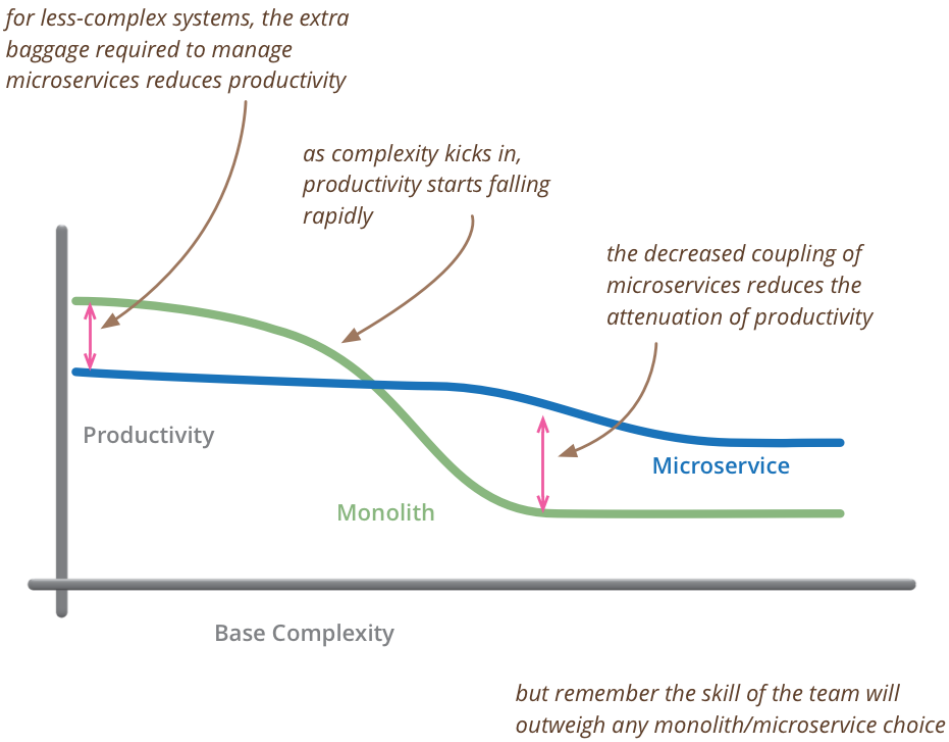
Інший наслідок використання сервісів як компонент - більш явний інтерфейс між ними. Більшість мов програмування не мають хорошого механізму для оголошення Published Interface. Часто тільки документація і дисципліна запобігають порушенням інкапсуляції компонентів. Сервіси дозволяють уникнути цього через використання явного механізму віддалених викликів.

Тим не менше, використання сервісів подібним чином має свої недоліки. Дистанційні виклики працюють повільніше, ніж виклики в рамках процесу, і тому API повинен бути менш деталізованим (coarser-grained), що часто призводить до незручності у використанні. Якщо вам потрібно змінити набір відповідностей між компонентами, зробити це складніше через те, що вам потрібно перетинати межі процесів.

У першому наближенні ми можемо спостерігати, що сервіси співвідносяться з процесами як один до одного. Насправді сервіс може містити безліч процесів, які завжди будуть розроблятися і розвиватися разом. Наприклад, процес додатку і процес бази даних, який використовує тільки ця програма.

Існує не дуже багато прикладів ринкових продуктів, що працюють за такою парадигмою, проте актуальні приклади мають багато чого спільного. Крос-фунціональні команди відповідають за побудову і функціонування кожного продукту і кожен продукт розбитий на кілька окремих сервісів, які спілкуються між собою через шину повідомлень.

Великі монолітні додатки теж можуть бути розбиті на модулі навколо бізнес потреб, хоча, зазвичай, це не відбувається. Безумовно, побудова великих монолітних додатків саме таким чином є безумовно доречною.



Основна проблема тут в тому, що такі додатки мають тенденцію до організації навколо занадто великої кількості контекстів. Якщо моноліт охоплює безліч контекстів, окремим членам команд стає занадто складно працювати з ними через їх великого розміру. Крім того, дотримання модульних кордонів в монолітному додатку потребує суттєвого дисципліни. Явно окреслені межі компонент мікросервісів спрощує підтримку цих кордонів[14].

Як результат загального використання мікросервісного стилю програмування варто відзначити, для надпростих задач, та мінімальних механізмів функціоналу, вимоги реалізації мікросервісів породжують надлишкові задачі. Проте такі задачі залишаються цілком виправданими у випадку, якщо межі та масштаби проекту не є чітко визначеними, адже залишає суттєвий простір для рефакторингу. Також основною зручністю в плані розробки стає можливість поєднання різних технологій та мов програмування, і відповідно більшої декомпозиції задач. Такі аспекти мають стати особливо доречними та ефективними для розробки масштабних учбових проектів, в рамках навчального процесу, оскільки декомпозиція функціоналу відбувається більш вільно, і задачі розробки можуть бути

більше легко адаптовані під потреби дисциплін, в тому числі, в процесі їх реалізації.

Ключовою особливістю мікросовервісної архітектури в контексті стеку технологій стає те, що архітектура його ніяк не обмежує. Поєднання мікросервісів покладається цілком на системні ресурси, а через їх широку різноманітність, переваги від одних іншим будуть надаватись виключно обумовлюючись специфікою задачі. Кожен окремий мікросервіс може бути реалізований з використанням будь якого стеку технологій, тому що це ніяк не впливає на загальну систему.

Порівняння продуктивності роботи системи для монолітної та мікросерівсної реалізацій не виконано в ході дипломного проекту, оскільки потребує створення суттєво більшої кількості експериментального функціоналу. В рамках теоретичних обрахунків та спекуляцій, можна зробити висновок, з приводу загальних тенденцій та властивостей архітектур в плані їх ефективності. Керування та організація роботи кожного з процесів мікросервісної архітектури суттєво підвищує використання системних ресурсів, але фундаментальна схильність до асинхронної роботи системи закладає в проект високий потенціал до оптимізації і руйнує будь-які обмеження до полегшення роботи сервісу шляхом посилення апаратної частини.

Саме як ідеологічний розвиток ідеї agile-у ми можемо взяти мікросервісний стиль програмування, адже така архітектура суттєво більш схильна до механізмів Continuous Integration та Continuous Deployment. Таким чином в процесі розробки, дійсно є можливість мати на кожному етапі робочий продукт. А за рахунок повної контейнеризації, система стає набагато більш платформо-незалежною і простою для портування. В контексті розробки учбових проектів її можливості практично довільної декомпозиції дозволяють долучати різні команди розробки до роботи над неймовірно масштабними задачами видаючи їм мінімальні незалежні задачі для рішення.

Отже, внаслідок проведеного дослідження стало зрозуміло, що потрібно зосереджувати увагу на проектуванні гнучкої та масштабованої архітектури розподілу на взаємопов'язані частини, що складатимуть єдину систему, тим самим звільняючись від проблеми масштабування та нестачі ресурсів.

Література:

1. Microservices. – Режим доступу: https://martinfowler.com/. – Дата доступу: 14.05.2017
2. *Balalaie A.,* The Philosophy of Microservice Architecture / Microservices Book. *Balalaie A., 14.04.2015 Birmingham. Режим доступа: microservicesbook.io.* / Дата доступу: 04.05.2017
3. Microservices: Five Architectural Constraints. – Режим доступу: http://www.nirmata.com/2015/02/02/microservices-five-architectural-constraints/. – Дата доступу: 20.05.2017
4. Ньюмен С. Створення мікросервісів / Сем Ньюмен. – (Питер). – (Бестселлеры O'Reilly), 2016-543 с.
5. Continuous Deployment: Strategies.– Режим доступу: https://www.javacodegeeks.com/2014/12/continuous-deployment-strategies.html /– Дата доступу: 18.04.2017