ISSN 2519-2310 (Online)

КОМП'ЮТЕРНІ НАУКИ ТА КІБЕРБЕЗПЕКА, 2024, № 2 (26)

COMPUTER SCIENCE AND CYBERSECURITY, 2024, Iss. 2 (26)

DOI: https://doi.org/10.26565/2519-2310-2024-2-04
УДК 004.41

ВИЗНАЧЕННЯ КРИТЕРІЇВ ВИКОРИСТАННЯ СЕРВІСНОЇ (SOA) ТА МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ (MSA) ПОБУДОВИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Олег Сєдашев[1], аспірант факультету математики і інформатики,
e-mail: oleh.siedashev@student.karazin.ua, ORCID: https://orcid.org/0009-0001-3259-3141
[1]Харківський національний університет імені В.Н. Каразіна,
майдан Свободи, 4, Харків, 61022, Україна

Рукопис надійшов 7 листопада 2024 р. Отримано після рецензування 8 грудня 2024 р.
Прийнято 22 грудня 2024 р.

Анотація: У сучасній розробці програмного забезпечення одним із ключових завдань є вибір відповідної архітектури для системи на ранніх етапах її проектування. Ця стаття розглядає два популярні підходи побудови програмного забезпечення: сервіс-орієнтованої архітектури (SOA) та мікросервісної архітектури (MSA). На основі аналізу архітектурних особливостей, переваг та недоліків цих підходів, досліджуються критерії, які впливають на вибір архітектурної моделі в залежності від специфіки системи. Мікросервісна архітектура, завдяки своїй незалежності та можливості швидкого масштабування, краще підходить для динамічних систем з високими вимогами до гнучкості. Сервісно-орієнтована архітектура, навпаки, орієнтована на централізоване управління сервісами через ESB (Enterprise Service Bus) і надає кращі можливості для інтеграції та повторного використання компонентів у великих корпоративних системах, та які не потребують частих змін функціоналу. Основна увага у статті приділяється розробці методу оцінки, який дозволить розробникам програмного забезпечення та системним інженерам на ранніх стадіях проектування визначити яку з архітектур, SOA чи MSA, доцільніше використовувати для конкретної системи. Враховуючи різні технічні та вимоги, метод виділяє ключові критерії на які слід звертати увагу при обранні архітектури програмного забезпечення додатку.

Ключові слова: *інформаційні системи, архітектура програмного забезпечення, сервіс-орієнтована архітектура, мікросервісна архітектура*

Як цитувати: Сєдашев О. Визначення критеріїв використання сервісної (SOA) та мікросервісної архітектури (MSA) побудови програмного забезпечення. *Комп'ютерні науки та кібербезпека*. 2024; № 2(26): С. 41–50. https://doi.org/10.26565/2519-2310-2024-2-04

In cites: Siedashev O. (2024). Determination of software architecture (SOA) and microservice architecture (MSA) usage criteria. *Computer Science and Cybersecurity.* 2(26): 41–50. https://doi.org/10.26565/2519- 2310-2024-2-04 (in Ukrainian)

Постанова проблеми

Проектування програмного забезпечення є одним з найважливішим етапів його розробки, потребує детального аналізу функціональних та нефункціональних вимог, а також обрання архітектури та технологій, які будуть використовуватися. Наразі існує безліч інструментів та описаних підходів рекомендованих для використання побудови додатків з нуля, але розвиток побудови програмного забезпечення починався з підходу використання «монолітної архітектури». Підхід, коли уся система є одним цілим, фізично розташований на одній машині та виконує усі операції як один процес. Через виклики масштабування, підтримки та оновлення системи цей підхід еволюціонував в сервіс-орієнтований підхід. У сервіс-орієнтованій архітектурі (SOA) система розділяється на окремі сервіси, які спілкуються між собою через спільні протоколи, часто з використанням шини (ESB). Це дозволило підвищити модульність і спрощувати інтеграцію різних частин системи. Мікросервісна архітектура (MSA) виникла з подальшою потребою у ще більшій гнучкості та незалежності сервісів. На відміну від SOA, де сервіси можуть бути більш тісно пов’язані, в MSA сервіси є дрібнішими і повністю незалежними один від одного. Кожен сервіс може мати свою базу даних і бути розгорнутим та масштабованим автономно. Цей підхід знижує складність підтримки системи та дозволяє швидше впроваджувати зміни, але вимагає більш складного управління і координації. Наразі не існує чіткого визначення, або критеріїв за якими слід обирати використання SOA чи MSA, та супутніх технологій на ранніх етапах проектування програмного забезпечення. Неправильний вибір без урахування усіх факторів може дуже негативно позначитись на усій системі та її здатності адаптуватися до змін бізнес-вимог у майбутньому або змін технологій.

Аналіз останніх досліджень і публікацій

Дослідження сервісного підходу побудови програмного забезпечення не зупиняються, та практично використовуються в індустрії. У роботі «Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith» [1] Сем Ньюман досліджує перехід від монолітних архітектур до мікросервісних, та занурюється в тему декомпозиції системи, як концепту. Книга складається з 5 глав, які описують опис мікросервісної архітектури, коли ця архітектура є актуальним рішенням, проблеми та технічні аспекти міграції з монолітної архітектури, та патерни, які допомагають вирішити ці виклики. В іншій роботі Сема Ньюмана «Building Microservices: Designing Fine-Grained Systems» [2] розглядаються загальні підходи архітектури мікросервісів, їх проектування та реалізація. Автор акцентує увагу на принципах автономності сервісів, управлінні даними та інтеграції, надаючи практичні рекомендації для створення мікросервісних систем. Ньюман також підкреслює важливість тестування та моніторингу для забезпечення стабільності та продуктивності системи. У персональному блозі [3] про мікросервісну архітектуру Chris Richardson дає визначення поняттям пов’язаним з мікросервісами та описує найкращі практики індустрії, які використовуються при побудові мікросервісних додатків. У книзі Mark Richards, Neal Ford «Fundamentals of Software Architecture: An Engineering Approach» [4] детально описується що таке архітектура програмного забезпечення, архітектурне мислення, поняття модульності, характеристики архітектури ПЗ, архітектурні стилі та патерни проектування. Зокрема розглядаються наступні стилі архітектури програмного забезпечення: монолітна, MVC (Model-View-Controller), Event-Driven Architecture, SOA, MSA. Особлива увага приділяється інженерним компромісам і рішенням, які виникають в процесу вибору між цими архітектурними рішеннями та надаються приклади оцінки архітектури програмного забезпечення, включаючи аналіз її структури, співвідношення сервісів до команд та практик розробки.

Мета статті

Метою статті є розробка методу оцінки, який дозволить на ранніх стадіях проектування програмного забезпечення визначити доцільність використання сервісно-орієнтованої архітектури (SOA) або мікросервісної архітектури (MSA). Визначення переваг та недоліків підходів, які використовуються сервісно-орієнтованою та мікросервісною архітектурою за допомогою порівняння ключових критеріїв. Це дозволить інженерам та розробникам обґрунтовано приймати рішення, щодо вибору архітектурного підходу, враховуючи вимоги до системи, її масштабованість, гнучкість, а також технічні та бізнесові критерії.

Основний матеріал

Монолітна архітектура є традиційним підходом до побудови програмного забезпечення, коли всі компоненти інтегровані в один єдиний виконуваний файл або додаток. Вона передбачає єдиний кодовий базис, де всі частини системи — від користувацького інтерфейсу до бази даних — об'єднані в одну цілісну структуру. Це робить систему більш керованою, оскільки всі зміни легко відслідковуються та контролюються, а процес розгортання є відносно простим. Такий підхід також спрощує тестування, оскільки весь додаток можна перевіряти як єдине ціле. Однак цей підхід має свої недоліки. Зі зростанням системи код стає важчим для підтримки, що ускладнює внесення змін і оновлень. Масштабування такої архітектури стає складним, оскільки потребує збільшення ресурсів на рівні всієї системи, навіть якщо лише одна її частина потребує додаткових ресурсів. Внесення змін в одну частину системи може вплинути на інші частини, що підвищує ризики та затримує розробку нових функцій. Таким чином, монолітна архітектура підходить для невеликих проектів або систем, що не потребують високої гнучкості та масштабованості, але може бути менш ефективною для великих і складних систем із постійно змінюваними вимогами [1, с. 11-49].

Перехід від монолітної архітектури до сервісно-орієнтованої архітектури (SOA) став важливим етапом у розвитку програмного забезпечення, оскільки дав змогу вирішити багато проблем, притаманних монолітним системам. Головними компонентами SOA архітектури є сервіси, Enterprise Service Bus (ESB), реєстр сервісів, контракти сервісів і композиція сервісів (Рис. 1). Сервіси є основними одиницями, що виконують конкретні бізнес-функції, взаємодіючи через стандартизовані інтерфейси. ESB відіграє роль центрального посередника для обміну повідомленнями між сервісами, відповідаючи за маршрутизацію та інтеграцію. Реєстр сервісів зберігає всі сервіси та їхні інтерфейси, дозволяючи іншим компонентам знаходити потрібні сервіси. Контракти сервісів визначають правила взаємодії між сервісами і клієнтами, описуючи формати даних та протоколи обміну. Композиція сервісів дає змогу об’єднувати кілька сервісів для виконання складних бізнес-процесів.

Переваги SOA включають гнучкість та адаптивність, оскільки зміни або додавання нових сервісів можуть бути виконані без впливу на інші компоненти. Масштабованість є ще однією сильною стороною SOA, дозволяючи кожному сервісу масштабуватися окремо, що забезпечує ефективне використання ресурсів. Архітектура підтримує повторне використання, оскільки сервіси можуть використовуватися в різних системах та бізнес-процесах. Також SOA забезпечує можливість інтеграції різних систем через стандартизовані протоколи взаємодії. Модульність SOA полегшує підтримку, тестування та розвиток системи. Недоліками SOA є складність впровадження, що може вимагати значних ресурсів для побудови необхідної інфраструктури. Високі витрати пов’язані із підтримкою ESB та інших компонентів, що робить архітектуру дорожчою у порівнянні з іншими підходами. Використання ESB може негативно вплинути на продуктивність, особливо у великих системах із високим трафіком даних між сервісами. Управління безпекою у SOA є складним завданням, оскільки сервіси взаємодіють

через мережу, що збільшує ризик безпекових інцидентів. Збої в роботі важливих компонентів, таких як ESB, можуть вплинути на всю систему, тому необхідно враховувати ці ризики при проектуванні архітектури.

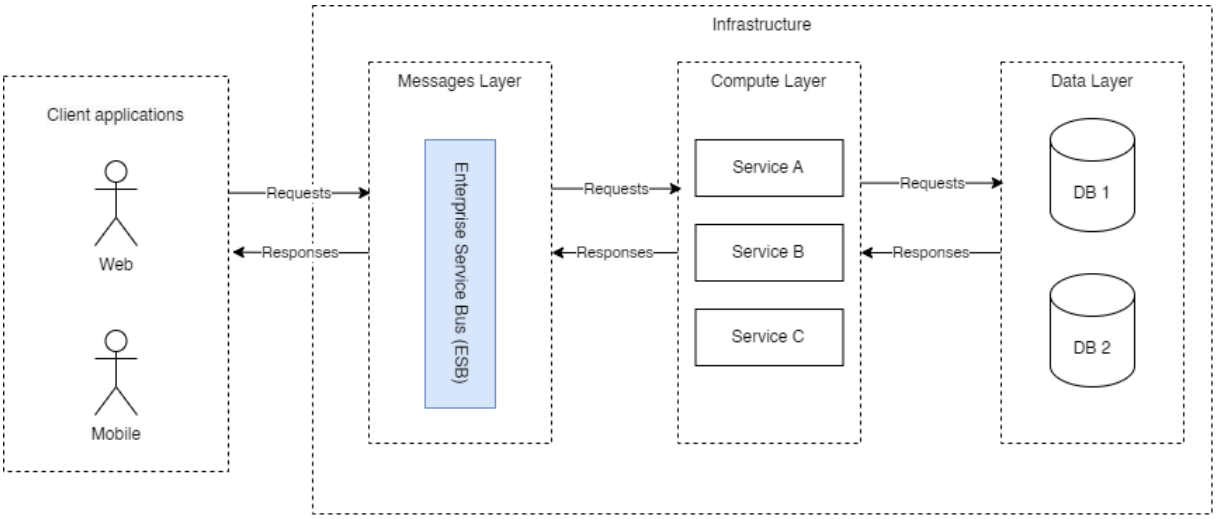

Рис. 1 – Сервіс-орієнтована архітектура (СОА)
Fig. 1 – Service-oriented architecture (SOA)

Порівняння монолітної архітектури та SOA за основними критеріями:

Таблиця 1. Порівняння монолітної та сервіс-орієнтованої архітектури (SOA)
Table 1. Comparison of monolithic and service-oriented architecture (SOA)

Критерій	**Монолітна архітектура**	**Сервіс-орієнтована архітектура (SOA)**
Структура	Єдиний кодовий базис, всі компоненти інтегровані в один додаток.	Система складається з окремих автономних сервісів.
Масштабованість	Масштабування всієї системи цілком, навіть якщо потрібно збільшити потужність тільки однієї частини.	Масштабування окремих сервісів незалежно один від одного.
Гнучкість	Зміни в одній частині можуть впливати на всю систему, що ускладнює гнучкість.	Зміни в одному сервісі не впливають на інші, що збільшує гнучкість.
Повторне використання	Компоненти складно використовувати повторно, оскільки вони тісно пов'язані між собою.	Сервіси можуть використовуватися повторно в інших системах або бізнес-процесах.
Затримка мережі (Latency)	Менша за рахунок відсутності мережевої взаємодії між компонентами.	Вища через необхідність мережевої взаємодії між сервісами.
Інтеграція	Інтеграція з іншими системами є складнішою та менш гнучкою.	Підтримка стандартизованих протоколів полегшує інтеграцію з іншими системами.

Підтримка та оновлення	Важко оновлювати, оскільки навіть незначні зміни можуть потребувати перезапуску всієї системи.	Оновлення одного сервісу не впливає на роботу інших, що спрощує підтримку.
Впровадження	Простіше впровадження для невеликих проектів або додатків.	Складніші вимоги до безпеки через взаємодію між сервісами через мережу.
Безпека	Безпека вбудована в межах одного додатку, але важче контролювати доступ до різних частин.	Складніші вимоги до безпеки через взаємодію між сервісами через мережу.
Управління відмовами	Відмова в одній частині може вплинути на всю систему.	Відмова одного сервісу не обов'язково впливає на інші, але потребує складного управління відмовами.
Модульність	Низька модульність, оскільки компоненти тісно пов’язані між собою.	Висока модульність, кожен сервіс автономний і має чітко визначені інтерфейси.
Розгортання	Розгортання всієї системи одночасно, що може бути проблемним у великих системах.	Можливість розгортання окремих сервісів незалежно один від одного.

Щодо взаємодії з рівнем даних у монолітній архітектурі взаємодія з даними відбувається через єдину спільну базу даних, що забезпечує високу швидкість доступу до даних, але призводить до тісної залежності між компонентами і складності масштабування. У SOA кожен сервіс є незалежним і має власну базу даних, а взаємодія з іншими сервісами відбувається через стандартизовані інтерфейси API. Це дозволяє кращу масштабованість і гнучкість, але ускладнює управління транзакціями та узгодженістю даних [1, с. 100-163].

Як висновок, монолітна архітектура добре підходить для невеликих або середніх проектів, де важлива швидкість розробки і простота управління системою. Але коли система зростає та необхідне обслуговування більшої кількості користувачів виникають проблеми з масштабуванням та підтримкою. SOA є еволюційним рішенням цих викликів та адресує їх краще ніж монолітний підхід, однак складність впровадження, управління розподіленими транзакціями та забезпечення узгодженості даних може стати проблемою. Також SOA вимагає більш складної інфраструктури та управління нею.

SOA це чудове архітектурне рішення для побудови середніх та великих додатків, саме завдяки використанню розподіленню функціональних вимог між сервісами та використанням модульності. Але коли виникає необхідність в ефективному, щодо використання ресурсів інфраструктури, масштабуванні або прискоренні розробки додатку, чи його компонентів, сервісної архітектури стає недостатньо. Еволюційно це призвело до появи мікросервісної архітектури (MSA), яка орієнтується на ще більш менший масштаб сервісів та їх функціональних обов’язків. Перехід від сервісно-орієнтованої архітектури (SOA) до мікросервісної архітектури (MSA) відбувся як відповідь на певні обмеження SOA та зростаючі потреби сучасних систем у більшій гнучкості, незалежності та швидкості розвитку. Виділяються наступні виклики:

1. Залежність від ESB. ESB стає вузьким місцем системи при появі більшої кількості сервісів, з якими потрібно комунікувати. Це призводить до проблем з ефективністю, керованістю та масштабуванні сервісів.
2. Незалежність розгортання. Хоча SOA пропонує модульність, сервісам зазвичай потрібна певна узгодженість при розгортанні, особливо через ESB. Це обмежує незалежне

оновлення або масштабування окремих сервісів.

3. Повторне використання та складність інтеграції. При потребі в повторному використанні сервісів, виникають додаткові виклики в інтеграції та забезпеченні сумісності між різними компонентами.

Мікросервісна архітектура (Рис. 2) — це підхід до проектування програмного забезпечення, при якому додаток розбивається на невеликі, незалежні сервіси, кожен з яких виконує окрему функцію і може розвиватися, розгортатися та масштабуватися автономно. Кожен мікросервіс функціонує як окрема одиниця, яка взаємодіє з іншими мікросервісами через легковагі протоколи, такі як HTTP, або за допомогою черг повідомлень (message queues) [2, с. 1-11].

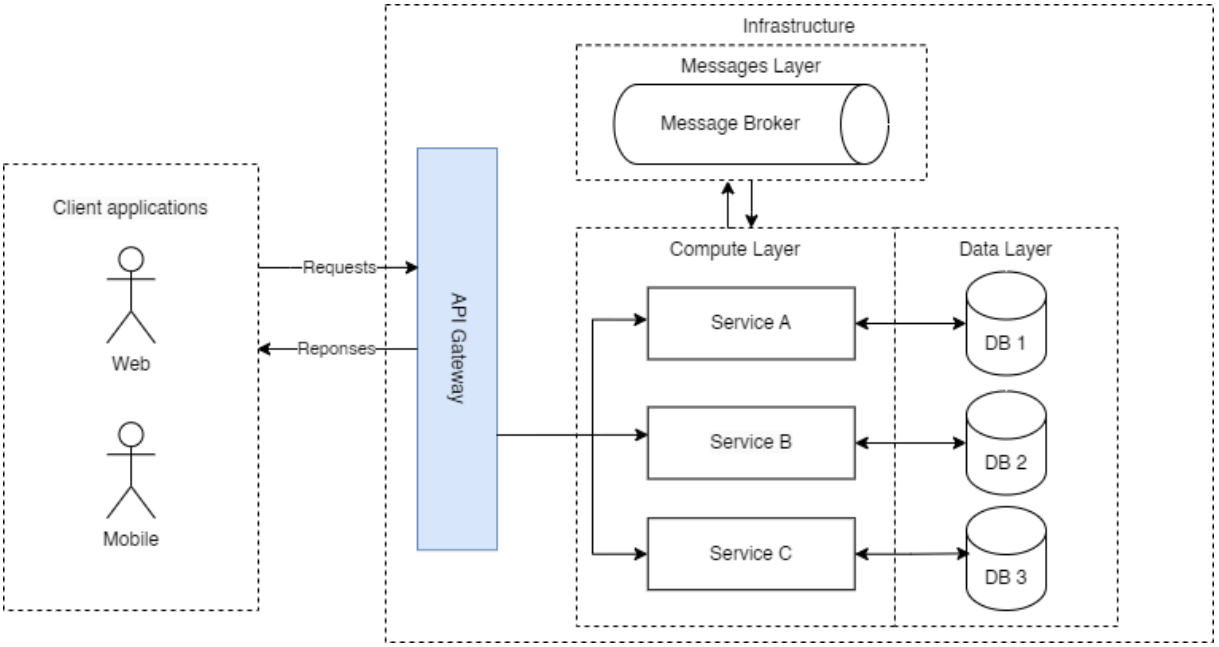

Рис. 2 – Мікросервісна архітектура (MCA)
Fig. 2. – Microservice architecture (MSA)

Основні принципи MSA [2, с. 245-250]:

1. Незалежність сервісів: Кожен мікросервіс є автономним, може працювати окремо від інших та мати власний життєвий цикл розробки.
2. Масштабованість: Мікросервіси можуть масштабуватися незалежно один від одного, що дозволяє оптимізувати використання ресурсів.
3. Мінімальні взаємозалежності: Мікросервіси повинні мати слабкі зв'язки між собою, що забезпечує легку змінюваність і оновлюваність системи.
4. Організаційна відповідність: Мікросервісна архітектура добре підходить для організацій з великими командами, які можуть працювати автономно над різними сервісами.

Порівняння MSA та SOA за основними критеріями [4, с. 235-274]:

Таблиця 2. Порівняння сервіс-орієнтованої архітектури (SOA)
та мікросервісної архітектури (MSA)
Table 2. Comparison of Service-Oriented Architecture (SOA) and Microservice Architecture (MSA)

Критерій	**Сервіс-орієнтована архітектура (SOA)**	**Мікросервісна архітектура (MSA)**
Зв'язність	Сервіси більш тісно інтегровані через ESB.	Слабко зв’язані, сервіси можуть функціонувати незалежно.
Комунікація між сервісами	Використання важких протоколів, таких як SOAP.	Використання легковагих протоколів, таких як HTTP/REST, gRPC.
Інфраструктура	Використання ESB для координації між сервісами.	Немає централізованої шини (ESB); сервіси самостійно керують взаємодією.
Масштабованість	Більш складне масштабування через централізовану структуру.	Легке масштабування кожного окремого сервісу.
Управління транзакціями	Централізоване управління через ESB.	Децентралізоване, кероване на рівні кожного мікросервісу.
Модульність	Менш модульна через інтеграцію з ESB.	Висока модульність; сервіси можуть бути легко змінені або замінені.
Швидкість розробки	Повільніший розвиток через необхідність координації між сервісами.	Швидка розробка, незалежне розгортання частин системи.
Повторне використання	Вищий рівень повторного використання сервісів.	Мікросервіси орієнтовані на вузькі завдання, що може знизити повторне використання.
Масштаб системи	Підходить для середніх і великих систем, але з меншою гнучкістю.	Найкраще підходить для великих, складних систем з частими оновленнями.

Виділяються наступні переваги MSA порівняно з SOA:

1. Децентралізація. Мікросервіси не залежать від централізованої шини (ESB) і спілкуються один з одним через легкі протоколи, такі як HTTP/REST або gRPC. Це дозволяє уникати вузьких місць, характерних для ESB в SOA, та знижує залежність між командами.
2. Масштабування. Кожен мікросервіс може бути масштабований окремо, що робить MSA більш гнучкою та легкою у підтримці. Це особливо корисно для систем з нерівномірним навантаженням, де окремі сервіси можуть мати різні вимоги до продуктивності.
3. Швидке розгортання. Кожен окремий мікросервіс є більш легким, можна розгорнути швидше. Це вирішує проблему координації, яка існувала в SOA, і дозволяє більш часті релізи та швидке реагування на зміни в бізнес-вимогах. Але не вирішує проблеми залежності комунікації між окремими сервісами.
4. Гнучкість технологій. У MSA кожен сервіс може використовувати різні технології та бази даних, що дає більшу свободу вибору інструментів для розв'язання конкретних завдань. У SOA сервіси часто обмежувалися єдиною технологічною платформою або потребували стандартизації для інтеграції.

Мікросервісна архітектура (MSA) стає кращим вибором у випадках, коли система повинна бути високо-гнучкою, потребує швидкого розвитку, незалежного розгортання та масштабування окремих сервісів, і готова до складнішої інфраструктури для підтримки розподілених компонентів. Ця архітектура добре підходить для сучасних хмарних додатків, з великими командами розробників, де кожна команда працює над окремим компонентом системи. Але водночас з’являються наступні фактори, які слід враховувати при обранні MSA:

1. Складність управління інфраструктурою: У порівнянні з монолітною архітектурою або SOA, MSA вимагає більш складної інфраструктури для оркестрації мікросервісів, управління контейнерами, моніторингу та налаштування мережевих взаємодій між сервісами. Це може вимагати значних витрат на підтримку та впровадження автоматизованих систем управління (Kubernetes).
2. Збільшення складності комунікації: У MSA сервіси взаємодіють через мережеві протоколи, що призводить до необхідності ефективного управління мережею, помилками комунікації та розподіленими транзакціями. Це може створити проблеми з латентністю та призвести до складних сценаріїв збоїв (наприклад, якщо один сервіс стає недоступним).
3. Проблеми з узгодженістю даних: Оскільки кожен мікросервіс може мати власну базу даних, управління узгодженістю даних між сервісами може бути складним. Мікросервіси не мають єдиної транзакційної системи, тому необхідно впроваджувати складні механізми узгодження (патерн SAGA, та інші) для забезпечення консистентності даних.
4. Збільшення обсягу мережевого трафіку: Кожен мікросервіс взаємодіє з іншими через мережу, що може призвести до значного збільшення мережевого трафіку. Це особливо критично для додатків, що вимагають високої пропускної здатності і швидкої обробки запитів.
5. Моніторинг і налагодження: Мікросервісна архітектура ускладнює моніторинг та налагодження системи. Необхідно відстежувати стан багатьох окремих компонентів, розподілених у різних середовищах, що вимагає впровадження спеціалізованих інструментів для логування, трасування запитів (distributed tracing) та моніторингу продуктивності.
6. Залежності між сервісами: Попри автономність мікросервісів, в складних системах може виникати сильна залежність між ними. Коли один сервіс виходить з ладу, це може призвести до каскадних збоїв у всій системі. Для запобігання таких проблем потрібні механізми резервування та стійкості до збоїв (circuit breaker patterns).
7. Проблеми з тестуванням: Тестування системи з багатьма мікросервісами може бути складнішим, оскільки потрібно враховувати різні сценарії інтеграції між ними, а також їхню взаємодію з іншими сервісами. Для цього потрібні комплексні методики автоматизованого тестування, що підвищує складність процесу тестування.
8. Витрати на підтримку інфраструктури: Використання мікросервісів може призвести до підвищених витрат на інфраструктуру, оскільки кожен сервіс потребує окремих ресурсів для розгортання, масштабування та моніторингу. Це також ускладнює управління обчислювальними та мережевими ресурсами.
9. Підвищені вимоги до команд розробки: Мікросервісна архітектура вимагає від команд розробки знання багатьох різних технологій, мікросервісних патернів та методів розподіленого програмування.

Висновки

Обидва підходи до побудови архітектури програмного забезпечення мають свої переваги й недоліки. Вибір між ними залежить від потреб бізнесу, специфіки проекту та вимог, щодо можливостей масштабування системи. Він повинен базуватися на аналізі конкретних потреб проекту, технічних обмежень та стратегічних цілей проекту або організації. SOA, краще підходить для більш структурованих систем, де важливо підтримувати стандартизацію, повторне використання сервісів та немає частих оновлень. Також для додатків, що потребують інтеграції зі старими системами і централізованого управління сервісами. MSA є ідеальним вибором для організацій, які прагнуть створювати високо-динамічні, масштабовані системи з частими оновленнями та гнучкістю в розробці, використовуючи хмарну чи подіну інфраструктуру.

Список літератури

1. Newman S. (2020) Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith 1st Edition. O'Reilly Media, 270 p. URL: https://dl.ebooksworld.ir/books/Monolith.to.Microservices.Sam.Newman.OReilly.9781492047841.EBooksWorld.ir.pdf
2. Newman S. (2015) Building Microservices: Designing Fine-Grained Systems 1st Edition. O'Reilly Media, 278 p.
3. Richardson C. Microservice Architecture. microservices.io. URL: https://microservices.io/
4. Richards M., Ford N. (2021) Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media, 419 p. URL: https://mrce.in/ebooks/Software-Fundamentals%20of%20Software%20Architecture.pdf
5. Mitra R., Nadareishvili I. (2020) Microservices: Up and Running: A Step-by-Step Guide to Building a Microservices Architecture 1st Edition. O'Reilly Media, 316 p.
6. C. Martin. R. (2018) Clean Architecture: A Craftsman's Guide to Software Structure and Design. Pearson, 432 p. URL: https://agorism.dev/book/software-architecture/%28Robert%20C.%20Martin%20Series%29%20Robert%20C.%20Martin%20-%20Clean%20Architecture_%20A%20Craftsman%E2%80%99s%20Guide%20to%20Software%20Structure%20and%20Design-Prentice%20Hall%20%282017%29.pdf
7. О.Ю. Ільїн та ін. Наукові записки УНДІЗ. Особливості розгортання мікросервісних додатків за допомогою системи керування контейнерами. 2019. №4(56). С. 49-60. DOI: https://doi.org/10.31673/2518-7678.2019.044960

DETERMINATION OF SOFTWARE ARCHITECTURE (SOA) AND MICROSERVICE ARCHITECTURE (MSA) USAGE CRITERIA

Oleh Siedashev[1], postgraduate student of the Faculty of Mathematics and Computer Science;
e-mail: oleh.siedashev@student.karazin.ua; ORCID: https://orcid.org/0009-0001-3259-3141
[1]*V. N. Karazin Kharkiv National University, Ukraine*

Manuscript was received November 7, 2024; Received after review December 8, 2024;
Accepted December 22, 2024

Abstract. In modern software development, one of the key tasks is to choose the appropriate architecture for the system in the early stages of its design. This article examines two popular software architecture approaches: service-oriented architecture (SOA) and microservice architecture (MSA). Based on the analysis of architectural features, advantages and disadvantages of these approaches, the criteria that influence the choice of an architectural model depending on the specifics of the system are

investigated. Microservice architecture, due to its independence and the possibility of rapid scaling, is better suited for dynamic systems with high requirements for flexibility. Service-oriented architecture, on the contrary, is focused on centralized management of services through ESB (Enterprise Service Bus) and provides better opportunities for integration and reuse of components in large corporate systems that do not require frequent changes in functionality. The main focus of the article is the development of an evaluation method that will allow software developers and system engineers to determine at the early design stages which of the architectures, SOA or MSA, is more appropriate to use for a specific system. Taking into account various technical and requirements, the method identifies key criteria that should be paid attention to when choosing an application software architecture.

Keywords: *information systems, software architecture, SOA, MSA*

References

1. Newman S. (2020) Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith 1st Edition. O'Reilly Media, 270 p. URL: https://dl.ebooksworld.ir/books/Monolith.to.Microservices.Sam.Newman.OReilly.9781492047841.EBooksWorld.ir.pdf
2. Newman S. (2015) Building Microservices: Designing Fine-Grained Systems 1st Edition. O'Reilly Media, 278 p.
3. Richardson C. Microservice Architecture. microservices.io. URL: https://microservices.io/
4. Richards M., Ford N. (2021) Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media, 419 p. URL: https://mrce.in/ebooks/Software-Fundamentals%20of%20Software%20Architecture.pdf
5. Mitra R., Nadareishvili I. (2020) Microservices: Up and Running: A Step-by-Step Guide to Building a Microservices Architecture 1st Edition. O'Reilly Media, 316 p.
6. C. Martin. R. (2018) Clean Architecture: A Craftsman's Guide to Software Structure and Design. Pearson, 432 p. URL: https://agorism.dev/book/software-architecture/%28Robert%20C.%20Martin%20Series%29%20Robert%20C.%20Martin%20-%20Clean%20Architecture_%20A%20Craftsman%E2%80%99s%20Guide%20to%20Software%20Structure%20and%20Design-Prentice%20Hall%20%282017%29.pdf
7. O.Yu. Ilyin et al. (2019) Scientific notes of the UNDIZ. *Features of deploying microservice applications using a container management system*. №4(56). P. 49-60. DOI: https://doi.org/10.31673/2518-7678.2019.044960 [in Ukrainian]