Архітектура Web Application

№ уроку: 36 **Курс:** Manual QA

Засоби навчання: Браузер, Microsoft Office

Огляд, мета та призначення уроку

Мета даного уроку – розглянути дві основні архітектури Web-застосунків та підходи для їх тестування.

Вивчивши матеріал даного заняття, учень зможе:

- Більш глибоко розуміти додаток, що тестується.
- Ефективно тестувати програми на мікро-сервісних та монолітних архітектурах.

Зміст уроку

- Монолітна архітектура
- Мікро-сервісна архітектура
- Підходи до тестування
    - Складності тестування мікро-сервісів
    - Складності тестування монолітних застосунків

Резюме

- **Моноліт** — у програмній інженерії монолітна модель відноситься до єдиної неподільної одиниці. Концепція монолітного програмного забезпечення полягає в тому, що різні компоненти програми поєднуються в одну програму на одній платформі. Зазвичай монолітний додаток складається з бази даних, клієнтського користувальницького інтерфейсу та серверної програми. Всі частини програмного забезпечення уніфіковані, і всі його функції керуються в одному місці.
- **Мікросервіси** — це тип сервісно-орієнтованої архітектури програмного забезпечення, орієнтований створення низки автономних компонентів, складових додаток. На відміну від монолітних програм, створених як єдине ціле, мікросервісні програми складаються з кількох незалежних компонентів, які склеєні разом за допомогою API.
- **Переваги монолітної архітектури**
    - **Спрощена розробка та розгортання.**
      Є багато інструментів, які можна інтегрувати для полегшення розробки. Крім того, всі дії виконуються з одним каталогом, що полегшує розгортання. Завдяки монолітному ядру розробникам не потрібно розгортати зміни чи оновлення окремо, оскільки вони можуть зробити це відразу і заощадити багато часу.
    - **Менше наскрізних проблем.**
      Більшість застосунків залежать від безлічі міжкомпонентних завдань, таких як контрольні журнали, ведення логів, обмеження швидкості і т. д. Монолітні програми набагато легше враховують ці питання завдяки своїй єдиній кодовій базі. До цих завдань простіше підключати компоненти, коли все працює в одному додатку.
    - **Найкраща продуктивність.**
      При правильній збірці монолітні програми зазвичай більш продуктивні, ніж програми на основі мікросервісів. Наприклад, програма з мікросервісною архітектурою може знадобитися виконати 40 викликів API для 40 різних

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

мікросервісів, щоб завантажити кожен екран, що, очевидно, призводить до зниження продуктивності. Монолітні програми, у свою чергу, забезпечують швидший зв'язок між програмними компонентами завдяки загальному коду та пам'яті.

- **Недоліки монолітної архітектури**
  - **Кодова база з часом стає громіздкою.**
    З часом більшість продуктів продовжують розроблятися і збільшуються обсягом, які структура стає розмитою. Кодова база починає виглядати дійсно громіздко і стає важкою для розуміння та зміни, особливо для нових розробників. Також стає все важче знаходити побічні ефекти та залежності. Зі зростанням кодової бази погіршується якість та перевантажується IDE.
  - **Складно впроваджувати нові технології.**
    Якщо у вашу програму необхідно додати якусь нову технологію, розробники можуть зіткнутися з перешкодами на шляху впровадження. Додавання нової технології означає переписування всієї програми, що є дорогим і потребує багато часу.
  - **Обмежена гнучкість.**
    У монолітних програмах кожне невелике оновлення вимагає повного повторного розгортання. Таким чином, усі розробники повинні чекати, поки це не буде зроблено. Коли кілька команд працюють над одним проектом, гнучкість може бути значно знижена.

- **Переваги мікросервісів**
  - **Висока відмовостійкість**
    При падінні одного з сервісів, решта залишаються в строю. Таким чином, неполадки в окремих сервісах не завадять всьому робочому процесу..
  - **Підвищена гнучкість**.
    Можна спробувати впровадити нову технологію малою кров'ю. Це буде значно швидше і при невдачі відкотити зміни просто. Змінюючи локально один із сервісів, ми не ризикуємо всією системою і час, який потрібний для змін, менший.
  - **Можливість масштабування по горизонталі**.
    Вертикальне масштабування (при використанні того ж програмного забезпечення, але на потужніших машинах) може бути обмежене пропускною здатністю кожного сервісу. Але горизонтальне масштабування (створення більшої кількості сервісів в одному пулі) не обмежене і може працювати з мікросервіс динамічно. Крім того, горизонтальне масштабування може бути повністю автоматизовано.
  - **Простота**
    Чим менше коду (а кожен окремий сервіс є цільною системою, тому не потрібно розбиратися у великій кількості деталей, що не стосуються цієї конкретної функції), тим простіше програмістам розібратися, що і як працює. До того ж, на це піде менше часу.
  - **Легко розробляти та розгортати**.
    Найбільша перевага мікросервісів перед іншими архітектурами полягає в тому, що невеликі окремі сервіси можуть створюватись, тестуватись та розгортатися незалежно. Оскільки одиниця розгортання невелика, це полегшує та прискорює розробку та реліз. Крім того, реліз однієї одиниці не обмежений випуском іншої, яка ще не завершена. І останній плюс тут полягає в тому, що ризики розгортання знижуються в міру того, як розробники релізують частини програмного забезпечення, а не цілий додаток.
- **Недоліки мікросервісів**
  - **Повідомлення між самими сервісами складне**.
    Оскільки кожен функціональний елемент ізольований, потрібна особлива

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

ретельність при побудові між ними грамотної комунікації, адже їм у будь-якому випадку доведеться обмінюватися запитами та відповідями один з одним. Зрозуміло, що зі збільшенням кількості сервісів складність у побудові їхнього повідомлення зростатиме.

- **Накладні витрати на комунікації між компонентами.**
  Виникають накладні витрати на комунікації між компонентами. Якщо в монолітному додатку модулі можуть спілкуватися через загальні змінні пам'яті, то мікросервіси надсилають один одному запити по мережі. Пересилання даних викликає невеликі затримки, які можуть бути критичними для застосунків, де важлива висока швидкість.
- **Складність організації даних між сервісами**
  Зростання числа послуг також тягне у себе зростання кількості баз даних, із якими ці послуги співвідносяться. Так само ускладнюється узгодженість даних та управління транзакціями, оскільки, на відміну від монолітної архітектури, мікросервіси використовують не одну загальну базу даних..
- **Ускладнюється управління інфраструктурою**.
  Кожен сервіс прийнято запускати на окремому сервері. Це вимагатиме від вас вкладень в автоматизацію роботи з серверами та надійного провайдера хмарних потужностей. Вартість розгортання хмарних машин вкрай низька, їх можна вводити в дію сотнями і тисячами буквально за секунди (звісно, якщо ваш хмарний оператор справді знає свою справу) — а це саме те, що життєво необхідне для гарної мікросервісної архітектури. Зробив нову версію мікросервісу – одразу створив під нього сервер.
- **Складності безпеки**.
  У мікросервісному додатку кожен функціонал, який взаємодіє через API ззовні, збільшує ймовірність атак. Ці атаки можуть відбутися лише в тому випадку, якщо під час створення програми не буде виконано належних заходів безпеки.
- **Підвищує планку рівня тех. компетенцій команди та керівництва**.
  Потрібен грамотний технічний посібник, який розуміє, що і як потрібно виділяти в мікросервіси і як організовувати комунікації. Фахівці з такими навичками не рідкість, але й не така вже часто знахідка на ринку айтішного персоналу.

Закріплення матеріалу

- У чому головна відмінність мікросервісів від моноліту?
- Що таке горизонтальне масштабування?
- У чому складність організації повідомлень у мікросервісах?
- Яка небезпека полягає у великих монолітних додатках, що розрослися?

Самостійна діяльність учня

Завдання 1
Знайдіть 5 компаній, які практикують мікросервісну архітектуру у своїх продуктах.

Завдання 2
Придумайте, як ви організували тестування мікросервісної архітектури. Ви можете скласти Тест План або записати будь-яким зручним для вас способом. Важливо врахувати усі види тестування.

Рекомендовані ресурси

Monolithic vs. Microservices Architecture
https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Microservices vs monolith: Which architecture is the best choice for your business?
https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business

Testing Strategies in Monolithic vs Microservices Architecture
https://www.browserstack.com/guide/testing-strategies-in-microservices-vs-monolithic-applications

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com