starts” і знижену вартість обслуговування контейнерів порівняно з викликами Lambda. Однак при низькому або хаотично змінному навантаженні серверлесс залишається економічно вигіднішим, та легшим у підтримці. ECS надає більший контроль для тонкої оптимізації, дозволяючи використовувати комбіновані метрики (довжина черги, CPU, кількість запитів), тоді як серверлесс не потребує ніяких налаштувань.

На основі проведених досліджень сформульовано такі висновки:

Використання ECS із авто скейлингом дозволяє покращити стабільність роботи додатків під великим навантаженням, прибрати довгі черги очікування процесингу даних, та дозволити їх асинхронну обробку. Загалом можна спостерігати зниження часу виконання задач у середньому на 40%. Загальні витрати на ресурси зменшуються на 30–50% для додатків із постійним високим навантаженням, через адаптацію під активність користувачів, на відміну від системи без скейлингу, де йде оплата під найгіршу ситуацію, або спостерігаються втрати швидкості виконання.

Lambda підходить для сценаріїв із нерегулярним трафіком або подій, але стає менш ефективним при високій інтенсивності запитів.

Додатково, бот із авто скейлингом забезпечує стабільну роботу для понад 100 користувачів без значних затримок, демонструючи переваги ECS для фінансових додатків.

Список джерел

1. Amazon Web Services. Amazon ECS Developer Guide. 2024.

2. Amazon Web Services. Amazon SQS Developer Guide. 2024.

3. AWS Architecture Blog. “Building Scalable Applications with ECS, SQS, and CloudWatch.”

4. Amazon Web Services. “Understanding Cold Starts in AWS Lambda: Implications for Real-Time Applications.”

5. Amazon Web Services. “Choosing Between AWS Lambda and Amazon ECS for Your Workloads.” AWS Compute Blog.

РОЗРОБКА ЗАСТОСУНКУ ДЛЯ АВТОМАТИЗОВАНОЇ ГЕНЕРАЦІЇ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ НА БАЗІ ПЛАТФОРМИ NODE.JS / DEVELOPMENT OF AN APPLICATION FOR AUTOMATED GENERATION OF MICROSERVICE ARCHITECTURE BASED ON THE NODE.JS PLATFORM

Колінько П.В./ Kolinko P.V.

Національний університет “Києво-Могилянська Академія”/ National university Kiev-Mohyla Academy

01054, Київ, вул. Григорія Сковороди, каф. мультимедійних систем,
тел. (044) 425-77-23, E-mail: pavlo.kolinko@ukma.edu.ua; факс (096) 988-59-39

The given works named “Development of an application for automated generation of microservice architecture based on the Node.js platform” explores how automated code generation and structured design streamline microservice-based application development. It introduces a custom software application built with Node.js, designed to automate the creation of microservice architecture through scaffolding. This method simplifies the setup of core structures, accelerating development. The article begins with a comparison between monolithic and microservice architectures. In monolithic systems, all components—user interface, business logic, and databases—are tightly integrated, making scaling and updates challenging as the system grows. Microservices break the application into independent services, allowing developers to

scale, test, and deploy each part individually. This flexibility leads to better fault tolerance and adaptability.

Одним з важливих питань, що постає на початку розробки бекенд застосунків, є вибір поміж монолітною та мікросервісною архітектурами, кожна з яких має свої переваги та недоліки [1]. Проте, варто зазначити, що у кожної з вищезазначених архітектур є спільний етап конфігурації, що передбачає налаштування середовища, встановлення бібліотек, підключення бази даних та багато супутніх кроків. У випадку розробки мікросервісного застосунку, процес конфігурації повторюється велику кількість разів, адже кожен сервіс є окремим застосунком в ізольованому середовищі, що значно сповільнює процес імплементації [2].

Для вирішення проблеми повільної розробки та запобіганню помилок у момент конфігурації використовують scaffolding-утиліти. Відповідно до назви scaffolding, цей підхід передбачає генерацію по якомусь загальному шаблону або “скелету”. Такий підхід необов’язково передбачає генерацію кінцевої версії продукту, що є готовим до використання, навпаки, частіше згенерована частина є лише основою, що спрощує та значно пришвидшує подальшу розробку застосунку. Для вирішення даної задачі, найбільш оптимальним рішенням є розробка застосунку для генерації мікросервісів, що є глобально незалежним від сторонніх бібліотек, є більш оптимальним і стабільним рішення, адже у цьому випадку, єдина технологія, що є необхідною для «життя» проєкту - це платформа Node.js. Ця платформа з часом все більше і більше набирає популярності, що означає її постійну підтри з боку розробників. Тому, кінцевим варіантом застосунку для генерації мікросервісів, стало об’єднання кращих практик з кожної дослідженої утиліти - простота у використання за допомогою CLI команд, та узагальнений широкий підхід для досягнення найвищого рівня абстракції.

Перед початком розробки власного scaffolding застосунка для генерації мікросервісів, необхідно визначитись з архітектурою. Потрібно створити максимально зрозумілий користувацький інтерфейс за допомогою CLI, в якому не буде потреби довго розбиратись. Було вирішено описувати всі необхідні конфігурації в одному джерелі - конфігураційному файлі. Опис конфігурацій у єдиному файлі запозичена з підходу Iac (англ. - Infrastructure as a Code). Iac (англ. – Infrastructure as a Code) [3], або інфраструктура як код, це підхід для опису та керування інфраструктурою хмарних сервісів за допомогою конфігураційних файлів.

Як мову програмування, було обрано Typescript. Розробка будь-якого програмного застосунку за допомогою Typescript має свій ряд переваг та недолік. З недоліків варто зазначити необхідність більшої витрати часу на розробку безпосередньо. Проте, для того застосунку як microgen (назва розробленого застосунку), критично важливим є правильний, односторонній потік даних та “виразний” опис кожної окремої сутності проєкту. Такий вибір мови програмування дозволяє підтримувати існуючий код, розширювати існуючий функціонал, мінімізує кількість помилок пов’язаних з неоднорідністю кодової бази. Для старту генерації, необхідно звернути увагу на консольні аргументи, що передаються під час ініціалізації, а саме змінну path. Оскільки скрипт має бути універсальним, і повинен дозволяти генерацію для будь-якої файлової системи, без прив’язки до конкретної директорії, необхідно надати

змогу користувачу динамічно зазначати абсолютне розташування microge.json файлу.

Організація роботи такого застосунку заснована по принципу переходу від найменшої задачі, до найбільшої. Такий підхід є необхідним, адже кожний наступний крок в генерації є фактично надбудовою над попереднім [4]. Перший етап передбачає парсинг та валідацію головного конфігураційного файлу microgen.json. Валідація файлу впроваджується окремою функцією, та є простою перевіркою, на відповідність даних заданому шаблону. Таким чином, відбувається перевірка, чи заповнені всі обов'язкові поля в загальній конфігурації проєкту або окремого мікрсервісу, чи існує імплементація обраного варіанту [4]. Генерація коду відбувається в рамках атомарного модуля, що представлена як окрема функції генерації. Для виконання консольних команд розроблено клас ChildProcessBuilder, що є імплементацію патерну Builder. За допомогою методу append клас об'єднує консольні команди в одну та зберігає їх у вигляді строки. Цей клас має на меті виконання ланцюга команд послідовно, оскільки процес створений методом exec є ізольованим [5] , і такий підхід - єдина можливість виконати набір команд послідовно. Вищевказана команда exec працює таким чином, що час на виконання команди є невизначеним, проте виклик цього самого методу виконується синхронно. Для вирішення даної проблеми було викростано пакет utils.promisify, котрий працює схоже до fs:promises і дозволяє обробляти проміси за допомогою async/await синтаксису, що дозволяє очікувати нового завершення виконання консольної команди.

З причини того, що проєкт написаний на мові програмування Typescript, який в свою чергу транспілюється, тобто перекладається в Javascript, всі файли, що мали розширення не .js або .ts не потрапляли в кінцевий Javascipt bundle, тобто побудований проєкт, готовий до використання. Тому був розрозроблений скрипт, що на етапі білда проєкту, копіювати всю директорію із Typescript директорії в аналогічну директорію Javascript білда.

В результаті розробки програмного забезпечення було досягнено основної мети по створенню стабільного застосунку, що є мінімально залежним від сторонніх бібліотек та має простий у використанні інтерфейс. Єдина "точка входу", що представлена конфігураційним файлом, дозволяє не тільки полегшити процес розробки на початкових етапах, а і легко керувати розширенням продукту, що використовує утиліту microgen. У випадку якщо бізнес вимоги до проєкту будуть розширені, та з'явиться необхідність створення нового мікросервісу, наприклад, засобами фреймворку Nest.js, додавши в конфігураційний microgen.json файл опис такого сервісу та повторно використавши CLI команду старту генерації, сервіс буде додано, а всі інші, що вже були згенеровані, не матимуть жодних змін, адже всі сервіси, що містять package.json файл будуть проігноровані для повторної генерації. Такий підхід є необхідним для запобігання програмного втручання в зміни внесені розробниками.

Розроблений додаток microgen має ряд вагомих переваг, таких як незалежність від бібліотек, що є нестабільними у підтримці, але і використовує вже розроблені рішення, такі як Nest CLI або Prisma CLI. Такий підхід дозволяє позбутись зайвої розробки додаткового коду та ресурсів застосунку, згенерувавши скелет мікросервісу, що використовує фреймворк Nest.js або ж ініціалізацію конфігураційного Prisma файлу за допомогою однієї команди без

необхідності власноруч прописувати код. Гнучка архітектура застосунку microgen дозволяє швидко додавати нові ресурси та розширювати функціонал.

Підсумовуючі, варто зазначити, що розроблена утиліта не має чітких відповідниках в рамках поставленої задачі. Вона спрощує та пришвидшує розробку в мікросервісів на базі платформи Node.js, що можуть бути згенеровані на базі найпопулярніших технологій цієї платформи. Проте, можливості застосунку обмежуються використанням невеликого переліку найпопулярніших технологій та відсутністю версіонування.

Список використаних джерел

1. Nagpal A. Monolithic vs Microservices Architecture: Advantages, Disadvantages, And Differences. Medium. URL: https://medium.com/@jasminepuno/monolithic-vs-microservices-architecture-advantages-disadvantages-and-differences-2bee6d1da8ca#:~:text=Monolithic%20architecture%20is%20a%20conventional,closely%20connected%20and%20centralized%20system.
2. Global Logic. Mikroservisna arkhitektura dlia pochatkivtsiv. Chastyna I.. GlobalLogic Ukraine. URL: https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/
3. Microsoft. What is infrastructure as code (IaC)? - Azure DevOps. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code.
4.Naik A. How to scaffold ExpressJS server and test it. Medium. URL: https://medium.com/craft-academy/how-to-scaffold-expressjs-server-and-test-it-d2a2ab1d30e0
5. INodeJS. Child process | Node.js v22.2.0 Documentation. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/api/child_process.html.

АНАЛІЗ ПАТЕРНІВ ПРОЄКТУВАННЯ У ВЕБРОЗРОБЦІ ТА ЇХ ЗАСТОСУВАННЯ У РОЗРОБЦІ ВЕБЗАСТОСУНКУ ДЛЯ АВТОМАТИЗАЦІЇ СТВОРЕННЯ РОЗКЛАДУ НАВЧАЛЬНОГО ЗАКЛАДУ / ANALYSIS OF DESIGN PATTERNS IN WEB DEVELOPMENT AND THEIR APPLICATION IN THE DEVELOPMENT OF A WEB APPLICATION TO AUTOMATE THE CREATION OF AN EDUCATIONAL INSTITUTION'S TIMETABLE

Борозенний С.О., Мисько Ю.М. / Borozennyi S.O., Mysko Y.M.
Національний університет «Києво-Могилянська академія» / National University of Kyiv-Mohyla Academy
04070, м. Київ, вул. Г. Сковороди, 2, кафедра мультимедійних систем, тел.: 044 425-77-53
E-mail: borozenyi@ukma.edu.ua

This work analyzes design patterns that can be used to create a web application. It also examines how different patterns can solve specific technical problems that developers face in their work. The result is a web application for creating a school timetable.

Сьогодні веброзробка давно перестала бути лише засобом для створення статичних вебсторінок наповнених текстовою інформацією та зображеннями. Натомість, все частіше можна зустріти складні вебзастосунки, здатні забезпечити широкий спектр послуг, як-от електронна комерція, соціальні мережі, графічні редактори та багато інших. Такі застосунки часто містять складну бізнес-логіку та інтеграції з різними сервісами, а отже мають відповідати високим вимогам до продуктивності та масштабованості.

Якщо раніше було достатньо простих скриптів для додавання динаміки на вебсторінку, то тепер, для забезпечення швидкої взаємодії з користувачем потрібна добре продумана та