дослідження треба розробити архітектуру центрального серверу, а також проаналізувати та визначити які принципи та підходи використовувати при розробці ядра локальної бази даних. Включаючи: контроль паралельності та транзакцій, рівень ізоляції, типи зв'язків, логування, парсер, інтерфейс запитів, архітектуру, модель та організацію сховища, оптимізатор та виконувач запитів.

Список використаних джерел

1. An overview on edge computing research / K. Cao et al. IEEE access. 2020. Vol. 8. P. 85714–85728. URL: https://doi.org/10.1109/access.2020.2991734.
2. Consistent local-first software: enforcing safety and invariants for local-first applications / M. Köhler et al. IEEE transactions on software engineering. 2024. P. 1–12. URL: https://doi.org/10.1109/tse.2024.3477723.
3. Gao Z. Design and implementation of embedded database security and reliability. Journal of physics: conference series. 2020. Vol. 1550. P. 032034. URL: https://doi.org/10.1088/1742-6596/1550/3/032034.

АВТО СКЕЙЛИНГ ВЕБ ДОДАТКІВ НА AWS З ВИКОРИСТАННЯМ КОНТЕЙНЕРІВ ТА СЕРВЕРЛЕСС ТЕХНОЛОГІЙ/AUTO SCALING WEB APPLICATIONS IN AWS USING CONTAINERS AND SERVERLESS

Шевчук В.І./Shevchuk V.I.

Київський Національний Університет Імені Тараса Шевченка / Taras Shevchenko National University of Kyiv

03127, м. Київ, проспект Голосіївський 130/57, тел. +38 093 363 67 45,

E-mail: vitshev120@gmail.com

This paper considers three scenarios of auto scaling in a cloud environment. Investigated pros and cons of those. In particular, regarding stability and cost.

Авто Скейлинг є ключовою технологією для оптимізації ресурсів і забезпечення стабільності веб додатків. У цій роботі розглядається практичне впровадження авто скейлингу на прикладі трьох сценаріїв використання AWS ECS, SQS, CloudWatch та Lambda .

Перший сценарій охоплює авто скейлинг для виконання асинхронних складних задач. Використання Amazon SQS дозволяє розподілити навантаження між задачами. CloudWatch метрики, зокрема кількість не оброблених повідомлень у черзі, використовуються для автоматичного збільшення кількості ECS-контейнерів під час пікового навантаження та їх зменшення у періоди спаду. На рисунку 1 представлено приклад графіку завантаження черги SQS та кількості контейнерів синім та помаранчевим кольорами відповідно.

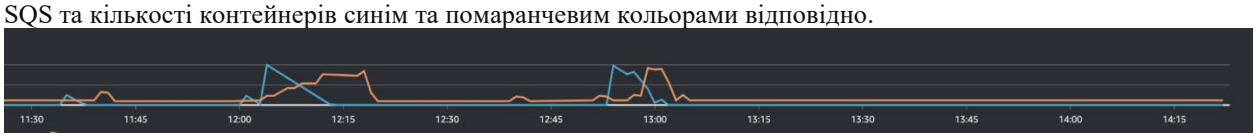

Рисунок 1. Графік завантаження черги SQS та кількості контейнерів

Другий сценарій демонструє авто скейлинг трейдинг бота для підтримки 10 активних користувачів на один контейнер. Контейнери автоматично додаються чи видаляються залежно від кількості запущених ботів для юзерів у реальному часі. Це рішення забезпечує стабільну роботу сервісу навіть при стрімкому зростанні активності користувачів, уникаючи перевантаження. Для роботи цього рішення було використано розподілення тасок у ECS по сервісам, з чітким завантаженням пам'яті та ресурсів на 10 контейнерів.

Третій сценарій стосується авто скейлингу API з використанням CloudWatch. Моніторинг таких метрик, як завантаження CPU та кількість запитів, дозволяє динамічно змінювати кількість ECS-контейнерів. Як альтернативу, можна було використати серверлес архітектуру (AWS Lambda).

Порівняємо ці рішення між собою. ECS із авто скейлингом демонструє кращу стабільність при високій інтенсивності запитів (>100 запитів на секунду), адже контейнери не потребують ініціалізації для кожного нового запиту. У той же час AWS Lambda забезпечує старт контейнерів в залежності від кількості запитів, ще може стати "вузьким місцем" при великих обсягах трафіку. ECS виграє за довготривалого навантаження через відсутність оплати за "cold

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