An improved approach to increase the fault tolerance of microservice software through automated functional and fault tolerance testing

Oleh Kuzmych
Institute of Computer Sciences and Information Technology
Lviv Polytechnic National University
Lviv, Ukraine
oleg.kuzmuch@gmail.com

Maksym Seniv
Institute of Computer Sciences and Information Technology
Lviv Polytechnic National University
Lviv, Ukraine
max1sudden@gmail.com

***Abstract*—Зростання вимог до якості, складність та критична важливість відмовостійкості в сучасних системах роблять необхідним впровадження нових і вдосконалених підходів автоматизованого тестування мікросервісного програмного забезпечення. У цій роботі на відміну від існуючих підходів, процес автоматизованого функціонального тестування адаптується до особливостей мікросервісних застосунків та інтегрується з тестуванням відмовостійкості, яке спрямоване на підвищення відмовостійкості — одного з ключових показників якості програмного забезпечення.**

Keywords—microservice architecture, reliability, fault tolerance, distributed systems, testing.

I. Introduction (*Heading 1*)

Протягом останніх десятиліть спостерігається стрімке зростання використання інформаційних технологій, значну роль у якому відіграє розвиток мережі Інтернет. Це, у свою чергу, призвело до різкого збільшення обсягів генерації та накопичення даних. Людство щодня продукує величезні масиви даних, які не тільки створюються, але й активно обробляються та зберігаються, що призводить до виникнення нових вимог до інформаційних систем.

Зростання потреби в обчислювальних потужностях і масштабованості програмного забезпечення стимулювало розвиток хмарних технологій. Їх поява сприяла створенню нових архітектур розробки програмного забезпечення, зокрема мікросервісної, що мала на меті підвищити надійність та гнучкість систем. Однак висока розподіленість мікросервісів створює значні проблеми з відмовостійкістю [1], оскільки відомо, що чим більше елементів у системі, тим більше можливих комбінацій дефектів, які можуть призвести до помилок [2].

.

Здавалося б вирішенням цієї проблеми мають слугувати підходи забезпечення відмовостійкості мікросервісного програмного забезпечення, проте в них є низка проблем. Деякі з цих проблем це недостатність автоматизації та ціна. Одним з рішень цих проблем може слугувати розроблення методу для підвищення відмовостійкості мікросервісного програмного забезпечення шляхом автоматизованого функціонального тестування та тестування відмовостійкості [3, 4].

Отже, існує велика потреба в удосконаленому підході для підвищення відмовостійкості мікросервісного програмного забезпечення шляхом безперервного функціонального тестування та тестування відмовостійкості, котрий на відміну від існуючих є адаптованим до мікросервісної архітектури, а також інтегрує процес тестування відмовостійкості в автоматизований процес функціонального тестування.

II. Related Works

Стрімкий розвиток хмарних технологій та перехід до нових розподілених архітектур програмного забезпечення, таких як мікросервіси та безсерверні обчислення, спричинили значні труднощі в забезпеченні відмовостійкості та тестуванні програмного забезпечення. Традиційні методи тестування, зокрема піраміда тестування, дедалі більше не відповідають вимогам складних сучасних архітектур. У відповідь на це з'являються нові підходи до автоматизованого функціонального тестування та забезпечення відмовостійкості в системах на основі мікросервісів.

У роботах [4, 5, 6] автори розглядають проблему забезпечення відмовостійкості мікросервісного програмного забезпечення. Вони класифікують відмови на різні типи і приходять до висновку, що для боротьби з ними потрібно використовувати різні методи забезпечення відмовостійкості.

У роботах [7, 8, 9] автори зосередились на патернах проектування, таких як Circuit Breakers, Bulkheads і Load Balancing, але виявили що використання тільки їх недостатньо для забезпечення відповідного рівня відмовостійкості, так як вони не допомагають відновитись системі після довготривалих збоїх. Також

автори звернули увагу на проблему несруктурованості великих даних, та те що їх легко втратити.

З іншого боку до проблеми підійшов автор роботи [10], він запропонував використовувати не один окремий метод для забезпечення відмовостійкості, а комплекс методів напрямлених на бородьбу з різними типами відмов. Також він зробив акцент що важливим етапом в забезпеченні відмовостійкості є етап тестування і його потрібно розвивати.

В іншому дослідженні [11] автори запропонували удосконалений підхід до автоматизованого інтеграційного тестування для безсерверного програмного забезпечення. У роботі обговорювались обмеження традиційних методів інтеграційного тестування, які часто є повільними, дорогими і не повністю автоматизованими. Автори запропонували концепцію сота тестування, яка адаптує традиційну піраміду тестування для більшої відповідності розподіленим архітектурам, таким як безсерверні та мікросервісні. Сота тестування надає пріоритет інтеграційному тестуванню над модульним, відображаючи зростаючу важливість забезпечення сумісності розподілених компонентів. У роботі також розрізняються локальні та глобальні інтеграційні тести, що дозволяє збалансувати швидкість, вартість і надійність тестування.

Низка науковців досліджує тестування [12, 13] зосереджується на вивченні впливу Chaos Engineering на архітектуру мікросервісів. Дослідження вказує на те, що впровадження Chaos Engineering дозволяє ідентифікувати слабкі місця у розподілених системах, що допомагає зменшити кількість збоїв і підвищити надійність сервісів. Додатково, автори розглянули різні інструменти для впровадження Chaos Engineering, зокрема Chaos Monkey, Chaos Mesh та інші, і зробили висновок про необхідність інтеграції таких підходів у процес безперервної інтеграції та розгортання (CI/CD).

Додатково до цього, у дослідженні [14] розглядається впровадження Chaos Engineering для мікросервісних архітектур, як методу покращення відмовостійкості. Chaos Engineering дозволяє виявляти слабкі місця та потенційні проблеми у системі ще до того, як вони стануть причиною серйозних збоїв у продуктивному середовищі. Це дослідження показує, що використання методик Chaos Engineering значно підвищує стійкість до збоїв, що особливо важливо в контексті мікросервісних архітектур, де кожен компонент системи може вплинути на роботу інших.

Спираючись на ці дослідження, у даній роботі пропонується подальша інтеграція тестування відмовостійкості у процес безперервного функціонального тестування. Це дозволить вирішити проблеми, виявлені в попередніх дослідженнях, та запропонувати комплексний підхід, який забезпечує не лише функціональну правильність мікросервісних додатків, але й їх стійкість до збоїв. Цей підхід є особливо важливим у контексті сучасних складних і розподілених систем, де вартість помилок може бути значною.

III. Опис удосконаленого підходу

A. Обґрунтування вибору підходу

Автоматизоване функціональне тестування є важливим аспектом забезпечення якості програмного забезпечення, особливо в умовах складних і розподілених архітектур, таких як мікросервіси. Особливої уваги заслуговує інтеграційне тестування так як дозволяє перевірити взаємодію між окремими компонентами системи, забезпечуючи їх коректну роботу як єдиного цілого, адже коли ці одиниці розглядаються окремо, вони функціонують правильно, майже без помилок, але коли їх об’єднують, вони можуть поводити себе непередбачувано та призводити до відмов. Тестування інтеграції має вирішальне значення, оскільки воно виконується на ранній стадії розробки та допомагає запобігти серйозним проблемам, які можуть виникнути пізніше.

Проте традиційні методи інтеграційного тестування часто стикаються з проблемами ефективності, швидкості та вартості, що особливо проявляється у великих системах.

У попередньому дослідженні [11] був запропонований удосконалений підхід до автоматизованого інтеграційного тестування, адаптований до безсерверних архітектур. Цей підхід базується на концепції соти тестування, яка дозволяє ефективно балансувати між швидкістю, вартістю та надійністю тестування, адаптуючи традиційну піраміду тестування до потреб сучасних архітектур.

Сота тестування передбачає розподіл тестів на локальні та глобальні інтеграційні тести, що забезпечує як швидку перевірку на початкових етапах розробки, так і точне тестування у реальному середовищі.

Цей підхід вже успішно використовувався для тестування у розподілених архітектурах, де ключовими аспектами є автоматизація, масштабованість і можливість адаптації до різних умов. Тому було вирішено вибрати його як основу для удосконаленого підходу для підвищення відмовостійкості мікросервісного програмного забезпечення шляхом безперервного функціонального тестування та тестування відмовостійкості

Для мікросервісних архітектур, де кожен сервіс є незалежним модулем, але взаємодіє з іншими через стандартизовані інтерфейси, цей підхід є особливо корисним. Він дозволяє ефективно перевіряти взаємодію між сервісами на різних рівнях, починаючи з локальних тестів і закінчуючи глобальними перевірками в реальному середовищі.

C. Покращення підходу для мікросервісів та інтеграція тестування відмовостійкості

Основним покращенням, яке пропонується в даному дослідженні, є інтеграція тестування відмовостійкості, та хаос-тестування, у процес автоматизованого функціонального тестування.

Тестування відмовостійкості - це процес перевірки здатності системи продовжувати працювати коректно навіть при виникненні несправностей або збоїв в її компонентах. Існують різні види тестування відмовостійкості, а саме хаос тестуання, інєкція помилок, тестування на перевантаження [15].

Також в майбутніх дослідженнях планується застосувати інструмент, який ідентифікуватиме причинно-наслідкові зв'язки у ланцюгах відмов мікросервісів (MacroHive). Це надаватиме більше інформації про збій, що дозволить швидше та ефективніше його усунути.

Відобразимо місце тестування відмовостійкості та хаос тестування в новій структурі під назвою пентагон тестування, що є модифікацією соти з попередньої роботи.

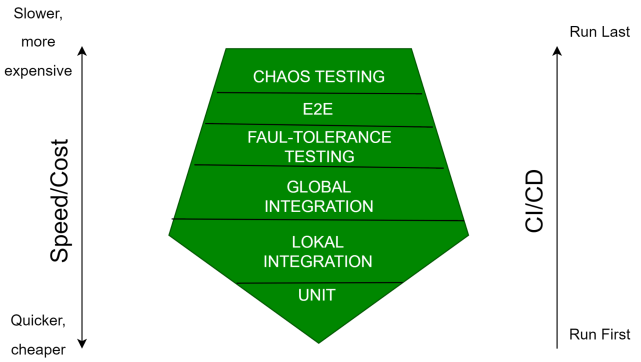

Рис. 2. Пентагон тестування

На рисунку 2 ми можемо побачити, що інтеграційні тести розділині на локальні інтеграційні тести та глобальні інтеграційні тести.

Локальні інтеграційні тести - це інтеграційні тести які будуть проводитись локально, а всі сервіси, які можливо, будуть емулюватись за допомогою спеціальних бібліотек. Тестування відбувається на рівні окремих сервісів для перевірки їх основних функцій та взаємодій з іншими сервісами. Ці тести є швидкими і виконуються першими в конвеєрі CI/CD

Глобальні інтеграційні тести - це інтеграційні тести які будуть проводитись в умовах близьких до реального середовища з використанням реальних даних і навантажень, що дозволяє перевірити коректність роботи сервісів у продуктивному середовищі.

Хаос-тестування: Після успішного виконання e2e тестів, група сервісів піддається хаос-тестуванню, яке включає симуляцію різних збоїв, таких як відмови сервісів, підвищення затримок або повна недоступність певних ресурсів. Це дозволяє оцінити стійкість системи та її здатність до відновлення.

Тестування відмовостійкості: перевіряє як працюють базові механізми забезпечення відмовостійкості (Повторні спроби, заслони, вимикачі ланцюга) в мікросервісах.

Таке розділення дозволить забезпечити дешевизну і швидкість при використанні локальних інтеграційних тестів і нададуть впевненість в результаті тестування при використанні глобальних інтеграційних тестів. Також дозволить виявляти помилки, які призводять до збоїв якомога швидше під час розробки мікросервісного програмного забезпечення.

Зараз в індустрії більшість програмного забезпечення створюється на нових розподілених архітектурах. Ці архітектури надали можливість розробляти частини програми різним командам майже незалежно одна від одної. Для такої розробки використовують можливості декількох середовищ (env, environment)

Ми в процесі розробки програми використовуватимемо чотири основні середовища.

DEV - середовище розробки

TEST - середовище тестування

PRE-PROD - середовище продакшену

PROD - середовище продакшену

Спираючись на таке розділення середовищ вкажемо в якому середовищі повинно відбуватись інтеграційне тестування та тестування відмовостійкості.

Оскільки локальне інтеграційне тестування швидке та недороге, то логічно його буде проводити так само часто як і модульне. Це дозволить виявляти помилки інтеграції ще на самому початку розробки сервісів. Для цього локальне інтеграційне тестування потрібно проводити перед розгортанням в середовище розробки. Такі інтеграційні тести будуть відповідати типу послідовних, оскільки в процесі розробки не всі сервіси є готовими і тому буде проводитись їх емуляція.

Глобальні інтеграційні тести будуть відбуватись в тестовому середовищі. Ці інтеграційні тести будуть відповідати типу інтеграційних тестів під назвою великий вибух, тому що більшість мікросервісів для тестування будуть реальними.

Тестування відмовостійкості ми проводимо після етапу глобальних інтеграційних тестів, щоб перевірити роботу базових механізмів забезпечення відмовостійкості (Повторні спроби, заслони, вимикачі ланцюга) в мікросервісах.

Хаос тестування виступає останнім етапом і відбувається після e2e тестування. Рандомним чином створюється ситуація відмови і логується як система поводить себе під час відмови. Чо змогла вона відновитись після відмову і за скільки часу

Тепер зобразимо місце цих та тестів відмовостійкості та хаос тестів в конвеєрах середовищ тестування та перед продакшену.

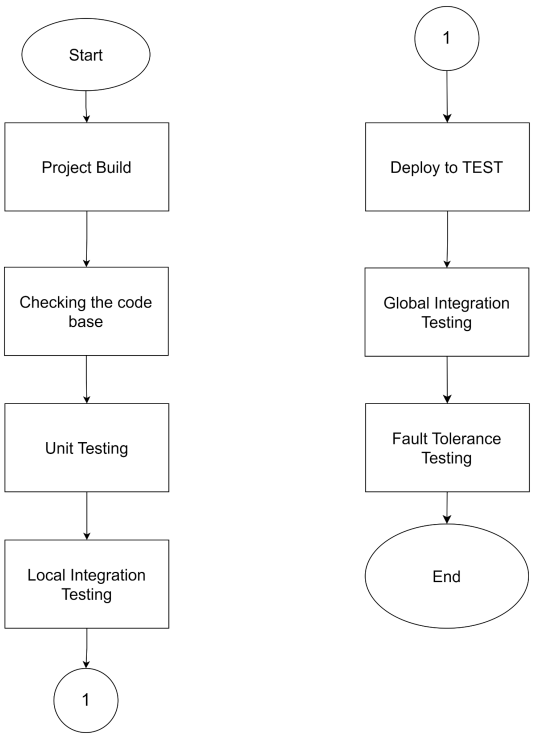

Рис. 3. CI середовища тест

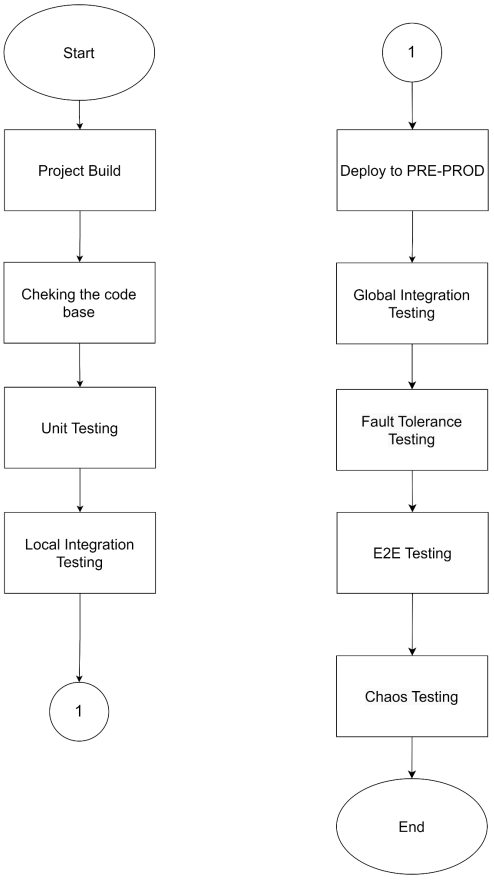

Рис. 4. CI середовища пре прод

D. Перевірка ефективності розробленого підходу

Визначимо які саме показники ми будемо вимірювати.

1. MTBF (Mean Time Between Failures) - підраховуємо загальний час роботи системи і ділимо його на кількість збоїв
2. Failure Rate - підраховуємо кількість збоїв за час тестуування і ділимо на загальний час роботи системи
3. Час тестування
4. Ціна тестування

Опис тестової програмної системи. Для оцінки ефективності розробленого підходу було обрано невелику мікросервісну програмну систему, на Python за допомогою фремворку FastAPI, що складається з 3 сервісів, які взаємодіють через REST API. Кожен сервіс виконує різні бізнес-функції, такі як обробка платежів, керування користувачами, управління замовленнями. Система використовує базу даних типу NoSQL, а саме MongoDB для збереження інформації та має компонент для управління чергами повідомлень для асинхронної комунікації між сервісами (AWS_SQS). Місцем розгортання системи було обрано хмару AWS.

Опис процедури тестування. Розробляємо систему з використанням соти тестування. Розгортаємо її в хмарі. Та на кожному етапі проводимо відповідні тести з соти тестування. Після цього виконуємо трьохденне стрес тестування для виявлення кількості помилок які не змогли відфільтрувати тести та вимірюємо наступні показники. (MTBF, Failure Rate та загальний час функціонального тестування.). Вдосконалюємо систему згідно до пентагону тестування. Розгортаємо нову систему в хмарі. Та на кожному етапі проводимо відповідні тести з пентагону тестування Проводимо те саме стрес тестування для вимірюваннння вказаних вище показників.

Продемонструємо отримані результати в таблиці:

Metric	Sota testing (3 days)	Pentagon Testing (3 days)
MTBF (minutes)	1440	2160
Failure Rate (відмови)	3	2
Час функціонального тестування (minutes)	67	88
Ціна тестування (USD)	0,134	0,176

Показники MTBF зріс на 50 відсотків та Failure Rata зменшився на 33 відсотка при використанні нашого підходу. Таке покращення повязане з тим, то інтеграція хаос тестування дозволила відловити більше помилок під час етапу тестування., збільшуючи час між відмовами та знижуючи частоту відмов.

Показник часу тестування при використанні нашого підходу збільшився на 31 відсоток. Таке збільшення повязане з кількістю тестів загалом та збільшенням їх видів. Інтеграційні тести розбиті на локальні та глобальні тести займають менше часу ніж просто інтеграційні, проте інші види тестів збільшують загальну тривалість тестування.,

Під ціною тестування ми маємо на увазі вартість CICD pipeline, що забезпечує автоматичний прогон тестів. Ддя цих систем ми використовували AWS CodePipeline. Зростання ціни тестування на 31% пояснюється наступними факторами: збільшення кількості тестів та часу тестування.

Ці показники не показують повної ефективнрості розробленого підходу, так як тестування відбувалось на малому проекті та тривало невеликий проміжок часу. В майбутньому планується застосувати цей підхід до великої системи та перевірити її ефективність в таких умовах.

IV. Conclusions

Продемонстровано потребу в удосконаленому підході для підвищення відмовостійкості мікросервісного програмного забезпечення шляхом автоматизованого функціонального тестування та тестування відмовостійкості.

Сформульовано цей підхід, де за основу було взято удосконалений підхід для інтеграційного тестування та концепцію соти тестування. Сота була модифікована в пентагон для відповідності потребам сучасних мікросервісних архітектур. Важливим аспектом є інтеграція хаос-тестування, що дозволяє оцінити стійкість системи до збоїв та підвищити її надійність.

Для перевірки ефективності сформульованого підходу було створено дві невеликі мікросервісні системи. Одна без використання розробленого підходу, а інша з. Для цих реалізацій було розраховано та проаналізовано декілька показників, MTBF та Failure Rata при використанні нашого підходу покращились на 50 та 33 відсотків відповідно. Показник часу тестування при використанні нашого підходу збільшився на 31 відсоток а ціна тестування збільшилась на 31 відсотки.

Отже, запропонований підхід є важливим кроком у напрямку підвищення відмовостійкості мікросервісного програмного забезпечення, і він має великий потенціал для застосування в різних галузях, де надійність і стабільність є ключовими вимогами, проте показники виміряні в цьому дослідженні не показують повної ефективнрості розробленого підходу, так як тестування відбувалось на малому проекті та тривало невеликий проміжок часу.

В майбутньому планується застосувати цей підхід до великої системи та перевірити її ефективність в таких умовах, а також планується розробка інших методів забезпечення відмовостійкості мікросервісного програмного забезпечення та їх апробація.

References

[1] ISO/IEC. (2023). Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model (ISO/IEC 25010:2023). https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-2:v1:en

[2] Ugrynovsky B. (2022). Methods and means of increasing the reliability of software, taking into account the process of its software aging. Ph.D. dissertation, Lviv Polytechnic National University. https://lpnu.ua/sites/default/files/2022/radaphd/19620/dissertation-uhrynovskyi-rc-6.pdf

[3] Baboi, M., Iftene, A., & Gîfu, D. (2019). Dynamic Microservices to Create Scalable and Fault Tolerance Architecture. Procedia Computer Science, Vol. 159, 1035–1044. https://doi.org/10.1016/j.procs.2019.09.271 4.

[4] Livora, T. (2016). Fault Tolerance in Microservices. Master's thesis, Masaryk University. https://is.muni.cz/th/ubkja/masters-thesis.pdf 5.

[5] Kumari, P., & Kaur, P. (2018, December). A survey of fault tolerance in cloud computing. Journal of King Saud University – Computer and Information Sciences, Vol. 33, Issue 10, 1159–1176. https://doi.org/10.1016/j.jksuci.2018.09.021

[6] Molchanov, H., & Zhmaiev, A. (2018). CIRCUIT BREAKER IN SYSTEMS BASED ON MICROSERVICES ARCHITECTURE. Advanced Information Systems, 2(4), 74–77. https://doi.org/10.20998/2522-9052.2018.4.13 8.

[7] Falahah, Surendro, K., & Sunindyo, W. D. (2021). Circuit Breaker in Microservices: State of the Art and Future Prospects. IOP Conference Series: Materials Science and Engineering, 1077(1), 012065. https://doi.org/10.1088/1757-899x/1077/1/012065 9.

[8] Mohammadian, V., Navimipour, N. J., Hosseinzadeh, M., & Darwesh, A. (2022). Fault-Tolerant Load Balancing in Cloud Computing: A Systematic Literature Review. IEEE Access, 10, 12714–12731. https://doi.org/10.1109/access.2021.313973

[9] Sambir A., Yakovyna V., Seniv M. Recruiting software architecture using user generated data // Perspective technologies and methods in MEMS design (MEMSTECH) : proceedings of XIIIth International conference, Polyana, April 20–23, 2017 - Львів : ПП "Вежа і Ко". – С. 161 – 163.

[10] Bouizem, Y. (2022). Fault tolerance in FaaS environments [Ph.D disertation, Université Rennes]. https://theses.hal.science/tel-03882666/file/BOUIZEM_Yasmina.pdf

[11] Kuzmych O. M., Lakhai V. Y., & Seniv M. M. (2023). Improved approach to automated software integration testing under the conditions of serverless architecture. Scientific Bulletin of UNFU, 33(3), 97–101. https://doi.org/10.36930/40330314

[12] Akuthota, Arunkumar. Chaos Engineering for Microservices // Master's Thesis, St. Cloud State University, 2023. https://repository.stcloudstate.edu/csit_etds/42

[13] Lakhai, V., Kuzmych, O., & Seniv, M. (2023). An improved method for increasing maintainability in terms of serverless architecture application. У 2023 IEEE 18th International Conference on Computer Science and Information Technologies (CSIT). IEEE. https://doi.org/10.1109/csit61576.2023.10324273

[14] Basiri, Ali, et al. Chaos Engineering // IEEE, 2023. https://doi.org/10.1016/j.jss.2023.03.012

[15] Dawson, S., Jahanian, F., Mitton, T., & Teck-Lee Tung. (б. д.). Testing of fault-tolerant and real-time distributed systems via protocol fault injection. У Annual Symposium on Fault Tolerant Computing. IEEE Comput. Soc. Press. https://doi.org/10.1109/ftcs.1996.534626