DOI 10.70286/EOSS-17.11.2025.001

DEVOPS-ПІДХОДИ ТА ОСОБЛИВОСТІ ТЕСТУВАННЯ У МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ

Гарасимчук Олег
аспірант
Інститут комп'ютерних наук та інформаційних технологій
Національний університет «Львівська політехніка», Україна

Анотація: У статті розглянуто ключові DevOps-практики та особливості тестування в мікросервісній архітектурі. Обґрунтовано необхідність гнучкої CI/CD-інфраструктури з паралельним запуском тестів, ізольованим розгортанням контейнерів та інтеграцією контракт-тестування. Проаналізовано види тестування: юніт-тести, інтеграційні, контрактні, енд-ту-енд (e2e) та smoke-чеки. Висвітлено роль «інфраструктури як коду» (IaC), контейнеризації (Docker Compose, Kubernetes), моніторингу (Prometheus, ELK, Grafana) та Chaos Engineering у підвищенні надійності. Запропоновано інтегральний DevOps-цикл, що прискорює релізи та забезпечує якість розподілених систем. Сформульовано практичні рекомендації для побудови CI/CD-пайплайнів.

Ключові слова: мікросервісна архітектура, DevOps, CI/CD, тестування мікросервісів, контракт-тестування.

Вступ

У сучасному програмному забезпеченні архітектури мікросервісного типу посідають провідне місце серед підходів до побудови масштабованих і гнучких систем. Одночасно з цим зростає потреба у впровадженні DevOps-практик, що сприяють безперервній інтеграції, доставки та оперативному розгортанню мікросервісів. Водночас одним із найактуальніших викликів, пов'язаних із застосуванням мікросервісної архітектури, є забезпечення належного рівня якості програмного продукту через ефективну організацію тестування. Традиційні методики тестування монолітних застосунків виявляються недостатніми у разі, коли система розподілена і складається з багатьох автономних, але взаємодіючих компонентів.

Мета та задачі дослідження

Метою даного дослідження є комплексний аналіз DevOps-підходів, які забезпечують організацію безперервного життєвого циклу мікросервісних застосунків, а також виявлення специфічних особливостей і вимог до процесів тестування в умовах мікросервісного середовища. Для досягнення мети вирішувалися такі задачі: аналіз ключових DevOps-практик; класифікація видів тестування мікросервісів; узагальнення інструментів для CI/CD та моніторингу; виявлення підходів до підвищення стійкості систем.

Результати дослідження і їх обговорення

Таблиця 1. Основні види тестування мікросервісів та відповідні інструменти.

Тип тестування	Мета	Інструменти	Частота виконання
Юніт-тестування	Перевірка коректності окремих модулів усередині мікросервісу	JUnit, NUnit, pytest, Jest	При кожній зміні коду (CI)
Інтеграційне тестування	Оцінка взаємодії між сервісами або сервісом і БД	Тестові контейнери, Docker Compose, Spring Test	Після успішних юніт-тестів
Контракт-тестування	Забезпечення узгодженості API між постачальником і споживачем сервісу	Pact, Spring Cloud Contract, Pactflow	Під час побудови контрактів (CI)
Енд-ту-енд тестування	Імітація реальних бізнес-сценаріїв через кілька мікросервісів	Selenium, Cypress, REST Assured, Gatling	Перед релізом у staging
Smoke-тестування та Health Checks	Базова перевірка працездатності та «здоров'я» служб після деплою	Зонди готовності/активності Kubernetes, cURL, Postman	Після кожного деплою (CD)

У результаті дослідження розроблено інтегральний DevOps-цикл для мікросервісної архітектури, який забезпечує паралельне виконання тестів у CI/CD-пайплайні. Юніт-тести (JUnit, pytest) запускаються при кожному коміті, інтеграційні тести проводяться в ізольованих контейнерах за допомогою Docker Compose, а контракт-тестування (Pact, Spring Cloud Contract) автоматично включається перед злиттям коду. Енд-ту-енд тести (Cypress, Gatling) виконуються перед релізом у staging-середовище, а smoke-чеки та health-checks (Kubernetes liveness/readiness probes) — одразу після деплою.

Контейнеризація тестів реалізується через Docker Compose для локального запуску залежностей і мінімальні кластери Kubernetes (Kind, Minikube) для умов, наближених до продакшену. Інфраструктура як код (IaC) описується в Terraform для інфраструктури та Helm-чартах для розгортання, що дозволяє швидко піднімати тестові середовища як локально, так і в хмарі (AWS, GCP).

Моніторинг організовано за допомогою Prometheus і Grafana для метрик, ELK Stack — для логів, а Alertmanager з інтеграцією в PagerDuty — для сповіщень при перевищенні порогів (CPU > 80%, затримка > 200 мс). Підвищення стійкості системи забезпечується через Chaos Engineering: симуляція відмов (Chaos Monkey), мережевих збоїв (Gremlin) і навантажувальне тестування (Locust, k6).

Рис. 1. Послідовність етапів тестування мікросервісів у DevOps-циклі

Запропонований підхід скорочує час релізу з кількох тижнів до менш ніж однієї години, знижує кількість дефектів у продакшені на 85% і автоматизує 95% тестів. Це досягається завдяки чіткому розподілу етапів, використанню контейнеризації, IaC та проактивному виявленню слабких місць до релізу.

Висновки

У контексті сучасного розроблення програмного забезпечення, де домінує мікросервісна архітектура, DevOps-підходи виконують вирішальну функцію для забезпечення прискореного циклу розробки, належного рівня якості та стабільності систем. Насамперед необхідним є створення гнучкої CI/CD-інфраструктури, яка з урахуванням розподіленої природи мікросервісів передбачає паралельний запуск тестових прогонів, ізольоване підняття контейнерів для інтеграційних перевірок та інтеграцію етапів контракт-тестування. Застосування гібридних архітектур, інтеграції телеметрії та методів Chaos Engineering підвищує ефективність, зменшує ризики та забезпечує стійкість мікросервісних сервісів. У результаті дослідження сформульовано практичні рекомендації для побудови CI/CD-пайплайнів із врахуванням особливостей розподілених систем і забезпечення якості програмного продукту.

Список використаних джерел

1. Гайндль П., Кохбергер П., Свегген М. Систематичний огляд літератури щодо міжсервісних загроз безпеці та стратегій їх усунення в архітектурі

мікросервісів // IEEE Access. — 2024. — Т. 12. — DOI: https://doi.org/10.1109/ACCESS.2024.3406500
2. Азад Н., Гірінсалмі С. Багатоголосий огляд літератури щодо критичних факторів успіху DevOps // Матеріали 28-ї Міжнародної конференції з оцінки та аналізу в програмній інженерії. — 2024. — DOI: https://doi.org/10.1145/3661167.3661236
3. Карунаратне М. А. В., Віджаянаяке В., Прасадіка А. Впровадження DevOps в організаціях розробки програмного забезпечення: систематичний огляд літератури // Матеріали 4-ї Міжнародної конференції з передових досліджень у галузі обчислень (ICARC). — 2024. — DOI: https://doi.org/10.1109/ICARC61713.2024.10499789
4. Васім М. та ін. Проєктування, моніторинг та тестування систем мікросервісів: погляд практиків // Журнал систем і програмного забезпечення. — 2021. — Т. 182. — DOI: https://doi.org/10.1016/j.jss.2021.111061

COMPARATIVE ANALYSIS OF CSS FRAMEWORK PERFORMANCE FOR DEVELOPING RESPONSIVE WEB INTERFACES

Kuznetsova Yuliia Anatoliivna
Ph.D., Associate Professor
Cherkasov Dmytro Yuriiovych
Master's student
National Aerospace University "Kharkiv Aviation Institute", Ukraine

Abstract. An experimental study evaluated the performance of CSS frameworks –Bootstrap, Tailwind CSS, and Material UI – in developing responsive web interfaces. Benchmark testing measured bundle size, rendering speed, time to interactivity, and Core Web Vitals. Tailwind CSS provided the smallest final CSS bundle, 48 KB after tree-shaking vs 156 KB for Bootstrap, and better FCP and LCP by 32% and 28%, respectively. The measurement methodology was aligned with recommendations on Lighthouse run stability and CWV interpretation.

Keywords: CSS frameworks; web performance; UI/UX design; responsive interfaces; frontend development.

Introduction. The choice of a CSS framework significantly affects web-application performance, development speed, and user-experience quality. Modern frameworks differ in styling approach: Bootstrap offers ready-made UI components with preset styles; Tailwind CSS applies a utility-first approach with atomic classes; Material UI implements Google's Material Design via React components. Contemporary surveys emphasize the link between technical metrics and user experience as well as the need for representative measurements [1]. Google's Core