УДК 004

Розбицький Р.Е., магістрант
Петросян Р.В., с.т. викладач
Державний університет «Житомирська політехніка»

ПОРІВНЯЛЬНИЙ АНАЛІЗ JAVASCRIPT-РАНТАЙМІВ У МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ

Порівняння JavaScript-рантаймів у контексті мікросервісної архітектури доцільно розглядати з позиції їхнього впливу на життєвий цикл сервісів, стабільність розподіленої системи, інтеграцію з інфраструктурою та експлуатаційні характеристики. У мікросервісній архітектурі середовище виконання прикладного коду виступає не лише технічним інструментом, а повноцінним елементом архітектури, який визначає швидкість запуску сервісів, ефективність обробки мережевих запитів, споживання ресурсів контейнерів та можливості інтеграції зі засобами моніторингу, логування і трасування [1].

Найбільш поширеним і фактично стандартним середовищем виконання JavaScript для серверних мікросервісних систем залишається Node.js. Його основною перевагою у контексті розподілених backend-систем є висока зрілість платформи, стабільність API та наявність розвиненої екосистеми бібліотек і фреймворків, орієнтованих саме на побудову серверних застосунків. Для мікросервісної архітектури принципово важливою є передбачувана поведінка рантайму під навантаженням, а також сумісність із типовими інструментами контейнеризації, оркестрації, централізованого логування та розподіленого трасування, що у випадку Node.js забезпечується багаторічною практикою використання у промислових середовищах [2]. Асинхронна подієво-орієнтована модель виконання Node.js добре узгоджується з характером мережевих мікросервісів, де значна частина часу витрачається на очікування відповідей від інших сервісів або зовнішніх ресурсів.

Разом з тим у контексті сучасних вимог до швидкості старту сервісів та ефективності використання ресурсів дедалі більшої уваги набувають альтернативні JavaScript-рантайми. Одним із таких середовищ є Deno, який орієнтований на усунення низки архітектурних обмежень, притаманних класичному підходу до виконання JavaScript на сервері. Для мікросервісної архітектури важливою особливістю Deno є вбудована модель безпеки, що передбачає явне надання дозволів на доступ до файлової системи, мережі та змінних середовища. Такий підхід потенційно дозволяє підвищити ізольованість окремих сервісів і зменшити ризики неконтрольованого доступу до ресурсів у розподілених середовищах [3]. Водночас у практичному застосуванні

для мікросервісних систем суттєвим обмеженням Deno залишається менша сумісність з існуючою серверною екосистемою та обмежена кількість бібліотек.

Окремий інтерес у контексті мікросервісної архітектури становить рантайм Bun, який позиціонується як високопродуктивне середовище виконання JavaScript і TypeScript із мінімальними накладними витратами. Для мікросервісних систем, що розгортаються у контейнеризованих середовищах і характеризуються частими перезапусками сервісів, важливою характеристикою є швидкість холодного старту та загальна швидкодія рантайму [4]. Саме у цих аспектах Bun демонструє відчутні переваги порівняно з класичним підходом, що може бути актуальним для допоміжних або високонавантажених сервісів. Однак у межах складних розподілених систем менша зрілість платформи та обмежений досвід експлуатації Bun у великих production-середовищах створюють додаткові ризики, пов'язані з довгостроковою стабільністю та підтримкою інфраструктурних інструментів.

У мікросервісній архітектурі вибір JavaScript-рантайму не може ґрунтуватися виключно на показниках продуктивності або швидкості запуску застосунків. Значно більшого значення набувають стабільність платформи, передбачуваність її поведінки у довготривалій експлуатації, наявність інструментів для централізованого логування, збору метрик і розподіленого трасування, а також сумісність із типовими процесами автоматизованого тестування та безперервного розгортання [5]. Саме ці характеристики визначають, наскільки безболісно конкретний рантайм може бути інтегрований у вже існуючу мікросервісну інфраструктуру.

Узагальнюючи результати порівняння, можна стверджувати, що Node.js на сьогодні залишається найбільш практичним і безпечним вибором для побудови промислових мікросервісних backend-систем завдяки своїй зрілості, екосистемі та широкій підтримці інфраструктурних інструментів.

Список використаної літератури:

1. Newman S. Building Microservices. 2nd ed. Sebastopol: O'Reilly Media, 2021. 352 c..
2. NGINX Inc. Microservices Reference Architecture: NGINX Whitepaper [Електронний ресурс]. 2021. Режим доступу: https://www.nginx.com
3. IBM Corporation. Microservices Guide 2023: Principles, Patterns, and Deployment Models [Електронний ресурс]. 2023. Режим доступу: https://www.ibm.com/cloud/architecture
4. Microsoft Azure Architecture Center. Microservices Architecture Style – Updated Best Practices 2022 [Електронний ресурс]. 2022. Режим доступу: https://learn.microsoft.com/azure/architecture
5. Dragoni N., Giallorenzo S., Lafuente A. L., et al. Microservices: Migration and Architectural Perspectives – 2020 Update. ACM Digital Library, 2020. DOI: 10.1145/3380768.3380775