УДК 004.7 **doi: 10.26906/SUNZ.2025.1.116-123**

С. В. Носко, С. С. Бульба, О. В. Коломійцев, Д. О. Лисиця, Г. І. Молчанов

Національний технічний університет «Харківський політехнічний університет», Україна, Харків

ПРОПОЗИЦІЇ ЩОДО АВТОРИЗАЦІЇ В САЙДКАР КОМПОНЕНТІ МІКРОСЕРВІСНОЇ АРХІТЕКРУРИ

Анотація. У статті розроблено пропозиції щодо доцільності реалізації авторизації у сайдкар компоненті мікросервісу, що дозволяє відокремити бізнес-логіку від завдань авторизації, ведення журналів, кешування тощо. Така реалізація дозволяє основним бізнес-модулям залишатися сфокусованими виключно на своїй бізнес-логіці, змінюючись лише тоді, коли є оновлення у бізнес-процедурах, а задачі, які є загальними майже для будь-якого сучасного рішення, передавати допоміжним компонентам таким як сайдкар. Правильна реалізація авторизації виступає ключовим аспектом будь-якої системи, є дуже важливим та вимагає врахування великої кількості особливостей та використання передових практик для забезпечення збереження даних від несанкціонованого доступу. Розглянуто можливості open-source рішення Keycloak. Таке рішення є дуже популярним сервісом для аутентифікації/авторизації, має підтримку стандартних протоколів ідентифікації, таких як: OpenID Connect, OAuth 2.0, SAML 2.0. Рішення підтримує багатофакторну аутентифікацію, підтримує контейнеризацію та може бути легко розподіленим у середовищі мікросервісів як Kubernates та інтегруватися з різними зовнішніми сервісами, включаючи Google, Facebook тощо. Розроблено архітектуру та пропозиції, що необхідні для виконання інтеграції з сайдкар компонентом. Проведено дослідження щодо продуктивності розробленого рішення. Отримано числові значення, що наведено у таблицях та на графіку.

Ключові слова: сайдкар, мікросервіс, авторизація, Keycloak, продуктивність, кеш, контейнер, Kubernates, масштабованість, open-source, Gatling, розподіл ресурсів, балансування навантаження, веб-сервер, інформаційна система.

Вступ

Постановка проблеми. В сучасних розподілених системах, що базуються на архітектурі мікросервісів, важливо забезпечити високу продуктивність, безпеку та масштабованість. Одним із викликів є ефективне управління авторизацією та аутентифікацією. Використання окремих мікросервісів для обробки авторизації може призвести до зниження продуктивності через дублювання зусиль та збільшення обсягу мережевих взаємодій.

Відповідальність мікросервісів за авторизацію додатково збільшує навантаження на них, а, отже, сповільнює реалізацію бізнес процесів. Підхід щодо авторизації має бути уніфікованим та централізованим, щоб спростити архітектуру рішення. Для реалізації таких комплексних завдань використовується сайдкар дизайн паттерн.

Авторизація та аутентифікація є одним із важливих процесів, де сайдкар знаходить ефективне застосування. До переваг такого рішення можливо віднести наступні:

– реалізація аутентифікації та авторизації у сайдкарі дозволяє управляти безпекою централізовано для усієї системи та усуває необхідність реалізації цих функцій у кожному мікросервісі окремо;

– використання сайдкару забезпечує однорідне застосування контролю доступу та політик безпеки для усіх мікросервісів, що усуває необхідність у додатковій конфігурації окремих мікросервісів, сприяє як зменшенню кількості похибок, так і полегшенню підтримці;

– масштабованість мікросервісів проходить простіше, оскільки усі налаштування вже інтегровані у сайдкар компонентах, а не у мікросервісах.

Авторизація у розподілених системах, особливо у архітектурах мікросервісів, є однією з найбільш складних та критичних задач, яка вимагає не тільки технічного, але й експертного дослідження у сфері керування безпекою. Складність полягає у забезпеченні надійної та безпечної авторизації без негативного впливу на продуктивність системи. Вибір неправильного рішення може призвести до вразливостей у безпеці та зниження ефективності системи.

Таким чином, необхідно обрати ефективний інструмент, який дозволить вирішити висвітлене наукове завдання. Одним із шляхів вирішення може бути застосування Keycloak, який активно використовується у індустрії для управління ідентичностями та авторизацією у розподілених системах.

Аналіз останніх досліджень і публікацій. Авторизація у мікросервісних архітектурах є ключовим компонентом, що впливає на безпеку та ефективність розробки і впровадження розподілених сервісів. У [1] проведено дослідження питання щодо безпеки мікросервісів. Визначено авторизацію як основний механізм для забезпечення дотримання принципів обмеженого доступу та ізоляції служб.

Документація Keycloak [2] надає інструкції щодо розгортання потужного серверу аутентифікації та авторизації у контексті мікросервісних архітектур, до якого сайдкар звертається для валідації токенів. Keycloak підтримує стандарти ID та доступу, такі як: OpenID Connect і OAuth 2.0, що дозволяє гнучко і безпечно управляти авторизацією користувачів.

У [3] проведено аналіз використання облікових даних API у мікросервісах, де пропонується підхід до ефективної авторизації та автентифікації через API шлюзи, що дозволяє покращити забезпечення безпеки у таких системах. Використовується API шлюз Kong, але його участь у процесах авторизації мінімальна, оскільки майже увесь об’єм роботи бере на себе сайдкар.

У [4] проведено дослідження різних підходів до механізмів авторизації у мікросервісах. Акцентуєть-

ся увага на використанні Role-Based Access Control (RBAC) як ефективного засобу управління доступом на різних рівнях. Їх робота підкреслює потребу у стандартизації взаємодій між незалежними сервісами, щоб забезпечити безпечний обмін інформацією. Такий підхід підтримується сервером Keycloak, який вибраний для реалізації у даній роботі.

У [5] проведено дослідження, як сайдкар може використовуватись для забезпечення безпечної авторизації, розділивши основні обов'язки між ядром сервісу та сайдкар проксі. Такий підхід дозволяє масштабувати авторизаційні контролі ефективніше та зменшити складність управління. Даний аналіз використовує схожі підходи у авторизації з використанням сайдкару як і у даній роботі.

У [6] проведено детальний аналіз використання Envoy Proxy як сайдкар рішення для управління авторизацією та моніторингу у мікросервісних архітектурах. Envoy дозволяє відстежувати та керувати забезпеченням безпеки на рівні кожного мікросервісу, подавати детальні логи доступу для аналізу та виявлення можливих атак на сервіс. Envoy Proxy це альтернатива рішенню, представленого у даній статті, але її архітектура не дозволяє кастомізувати та додавати нових можливостей таких як інтеграція зі штучним інтелектом.

Мета статті – розробка пропозицій щодо реалізації авторизації у сайдкар компоненті мікросервісу для відокремлення бізнес-логіки мікросервісу від завдань авторизації та підвищення продуктивності рішення.

Виклад основного матеріалу

Застосування підходу з сайдкарами і Keycloak у якості авторизаційного сервера дозволяє в цілому підвищити безпеку та управління доступом у мікросервісних архітектурах.

Keycloak має відкритий вихідний код, що надає обширні можливості для кастомізації та розширення функціональності для потреб бізнесу. Платформа адаптована для роботи у великих організаціях та здатна легко масштабуватися зі зростанням системи.

При розробці використовувалась версія, яка має інтеграцію з OpenID Connect Provider, що представлено на рис. 1.

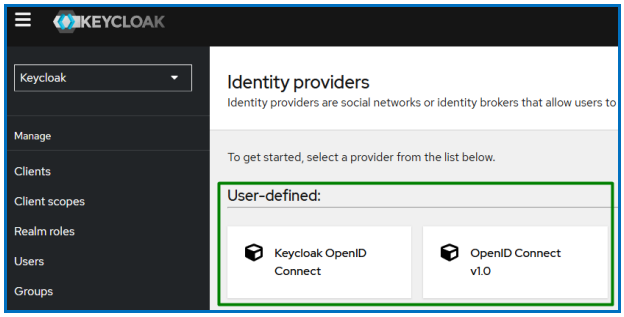

Рис. 1. Вибір OpenID Identity провайдера

Недоліком рішення є ускладнення управлінням системою при впровадженні додаткового компонента, що може виявитися складним у налаштуванні для людей без відповідного досвіду та вимагати додаткових обчислювальних ресурсів. Також, Keycloak є відносно комплексною системою, що потребує регулярних оновлень безпеки та підтримки.

Незважаючи на можливі ускладнення в управлінні та вимоги до ресурсів, переваги використання Keycloak як рішення для розумного сайдкара, особливо у контексті забезпечення безпеки, автентифікації та централізованого управління доступом, значно перевищують потенційні недоліки. Таке рішення забезпечує високий рівень безпеки та ефективності для мікросервісних архітектур [7, 8].

Використання Keycloak як окремого контейнера у Kubernetes середовищі є одним із оптимальних способів розгортання для досягнення високої доступності, масштабованості та управління. Таке рішення надає ряд важливих переваг:

– розгортання Keycloak у окремому контейнері забезпечує ізоляцію від інших частин системи, що підвищує безпеку та стабільність;

– Kubernetes підтримує високу доступність шляхом автоматичного перезапуску контейнерів у випадку їх падіння або помилок, забезпечуючи таким чином високу відновлюваність.

У якості типу контролера у Kubernates для Keycloak обрано StatefulSet замість Deployment, який використовується для stateless мікросервісів. Таким чином зберігається стан сесій, що забезпечує відновлення після збою без втрати даних як представлено на рис. 2.

У процесі реалізації системи управління доступом та ідентифікацією за допомогою Keycloak важливо враховувати інтеграцію та синхронізацію даних користувачів та їх ролей. Варіанти рішення цього завдання залежать від архітектурних особливостей вже існуючої системи.

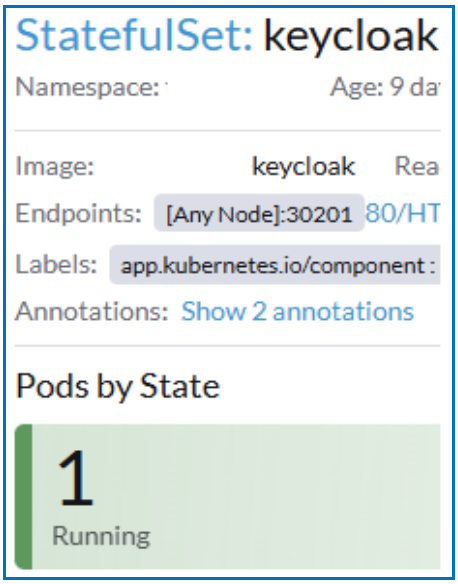

Рис. 2. Використання StetefulSet контролера

У контексті мікросервісної архітектури, де вже існує повнофункціональне зберігання користувачів та їх ролей, ефективним рішенням є використання Keycloak REST API для забезпечення синхронізації

даних. Такий підхід передбачає, що при зміні інформації про користувача у основній мікросервісній системі, такій як модуль mod-users, потрібно виконувати відповідні POST, PUT або DELETE HTTP запити до Keycloak API, що дозволяє забезпечити консистентність даних між різними компонентами системи.

На рис. 3 представлено список користувачів, зареєстрованих у визначеному Realm системи Keycloak. У даному контексті, Realm є логічним простором у системі, що містить окремі користувацькі ідентичності, облікові дані, ролі та інші конфігураційні параметри.

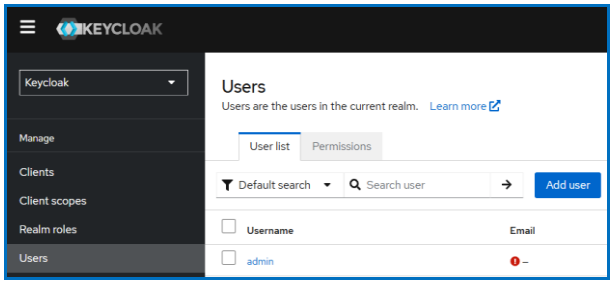

Рис. 3. Користувачі у системі

На рис. 4 представлені ролі, які доступні у системі для подальшого присвоєння користувачам. Ролі у контексті Keycloak необхідні для управління правами доступу, вони можуть бути як глобальними, так і визначеними для конкретного Realm.

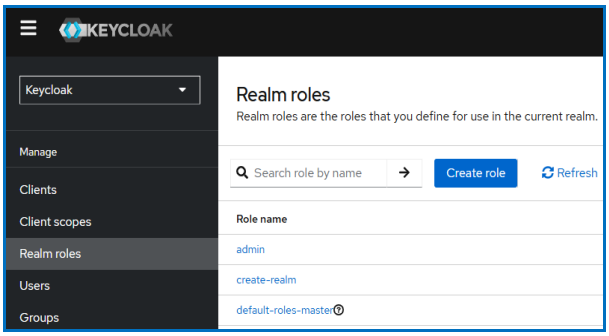

Рис. 4. Ролі у системі

Запропоновано у реалізації розумного сайдкару авторизацію здійснювати у сайдкар компонентах. Виклик між модулями здійснювати через відповідні сайдкари.

До інших варіантів реалізації запропонованих пропозицій можливо віднести наступне:

– централізований API шлюз. У даному випадку усі перевірки авторизації відбуваються перед тим, як трафік досягне мікросервісів. Даний шлюз може керувати аутентифікацією, шифруванням, логуванням та іншими аспектами безпеки для усіх вхідних запитів;

– обробка JWT токенів на рівні мікросервісів. Мікросервіси можуть самостійно обробляти JWT токени, декодуючи їх та перевіряючи валідність. При такій реалізації кожен мікросервіс повинен мати логіку для валідації токену, що призводить до дублювання логіки та ускладнення впровадження змін у способі авторизації;

– використання зовнішньої системи для оцінки політик, контролю та безпеки доступу таких як Open Policy Agent. Такі системи дозволяють централізувати правила авторизації та легко їх оновлювати.

У сучасних мікросервісних архітектурах, де досягнення високого рівня безпеки доступу є одним із ключових викликів, дослідження засобів авторизації критичне для забезпечення загальної безпеки системи. Використання «розумного» сайдкара для обробки авторизації, зокрема через інтеграцію із Keycloak, надає певні переваги порівняно з авторизацією, виконаною безпосередньо у мікросервісі.

Перш за все, такий підхід сприяє відокремленню зон відповідальності. Сайдкари беруть на себе повну відповідальність за обробку авторизації, звільняючи основний контейнер мікросервісу від необхідності включати розгалужені перевірки безпеки, які можуть з часом змінюватись або оновлюватися.

Додатково, використання уніфікованої логіки авторизації у сайдкарі сприяє виключенню дублювання логіки авторизації у кожному мікросервісі, значно спрощуючи процес розробки та підтримки. При змінах у правилах авторизації, потрібно здійснити оновлення лише в одному місці – сайдкарі, що зменшує комплексність управління оновленнями, порівняно з необхідністю різних адаптацій у кожному мікросервісі.

Використання сайдкара, також, підвищує загальний рівень безпеки, оскільки зменшує ймовірність помилок у реалізації авторизації через меншу кількість точок входу та стандартизацію підходів.

Кроки взаємодії Keycloak з мікросервісною архітектурою з використанням "smart sidecar" для обробки авторизації виглядають наступним чином:

1. Користувач взаємодіє з інтерфейсом користувача (UI), наприклад, веб-сторінка, який викликає відповідний API ендпоінт. HTTP-запит, здійснений користувачем, містить у своєму заголовку токен авторизації (JWT токен).

2. Запит спочатку обробляється Kong, який виступає у ролі API Gateway. Kong аналізує HTTP заголовок, визначає URI запиту та, використовуючи внутрішню схему бази даних, де збережено відповідності URI до специфічних мікросервісів, перенаправляє запит до відповідного мікросервісу.

3. Запит досягає відповідного мікросервісу та перехоплюється сайдкаром, пов'язаним із цим мікросервісом. Сайдкар відіграє ключову роль у обробці запиту на цьому етапі.

4. Сайдкар здійснює запит до Keycloak, передавши необхідні дані про користувача (зазвичай у вигляді вищезгаданого JWT токену). Запит включає важливі контекстні дані.

5. Keycloak перш за все перевіряє чинність самого токена – зокрема, правильність його формату, підпису, а також час дії (expiration).

6. Після успішної валідації JWT токена Keycloak додатково перевіряє, чи має користувач достатні права авторизації для виконання запиту. Це включає перевірку заданих у токені ролей та дозволів користувача порівнюючи з обмеженями доступу, визначених у Keycloak для даного ендпоінту або ресурсу.

На рис. 5 показана перша частина діаграми авторизації, де запит обробляється компонентом Kong, що виступає у ролі Api Gateway. На рис. 6 показана друга частина діаграми, де сайдкар викликає сервер авторизації Keycloak для перевірки прав користувача перед викликом свого мікросервісу.

Використання сайдкар-компонентів забезпечує можливість локалізації процесів авторизації на рівні окремого мікросервісу, що підвищує продуктивність системи, оскільки після успішної авторизації в Keycloak, сайдкар мікросервісу 1 може безпосередньо комунікувати з мікросервісом 2, оминувши повторні перевірки авторизації.

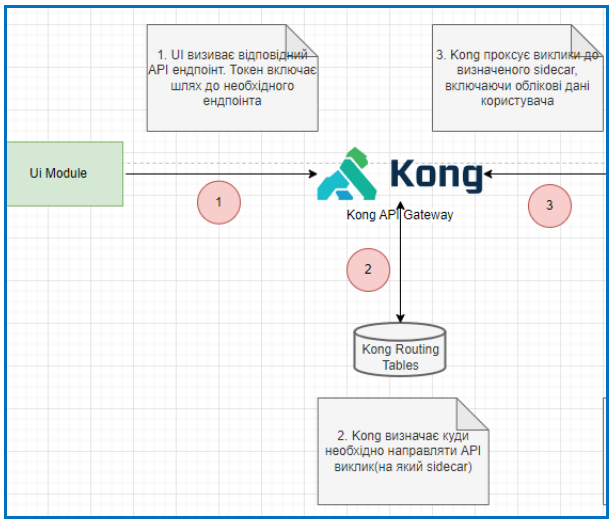

Рис. 5. Обробка запита Api Gateway

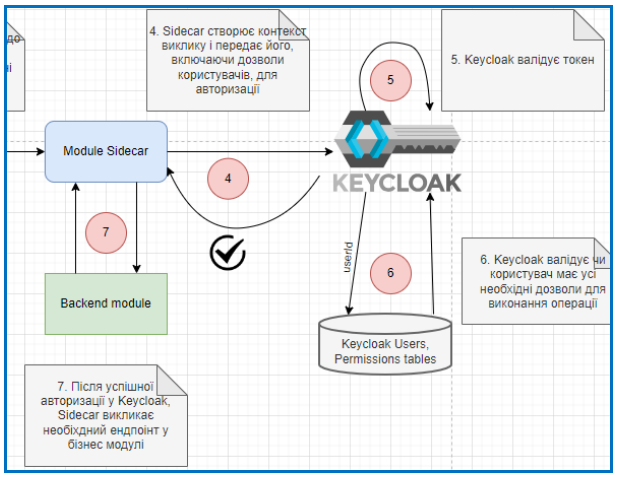

Рис. 6. Комунікація між сайдкаром і Keycloak

Однак, необхідність виконання авторизації для кожного API-виклику всередині сайдкара може створити значне навантаження на Keycloak, особливо у системах з високою частотою запитів. Таке навантаження може призвести до зниження загальної продуктивності авторизаційного Keycloak сервісу, що, відповідно, негативно вплине на ефективність роботи мікросервісної архітектури.

На рис. 7 представлено кількісні дані логування, які ілюструють звернення мікросервісів до відповідних сайдкарів для валідації токенів у Keycloak протягом визначеного 17-хвилинного інтервалу.

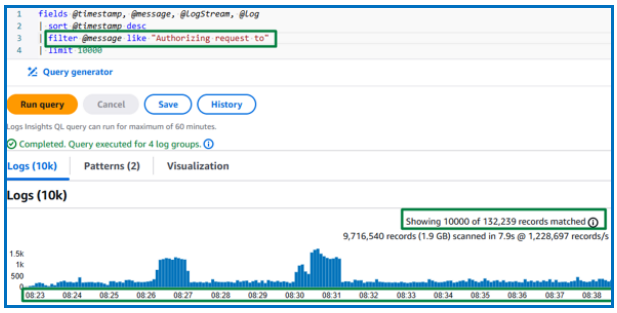

Рис. 7. Кількість звернення сайдкару до Keycloak під час пікових навантажень

Дані отримані з проведених випробувань стійкості системи, яка була розроблена.

З рис. 7 можливо стверджувати, що протягом вказаного періоду сайдкари здійснили 132,239 звернень до сервера автентифікації Keycloak. Дана статистика свідчить про значне навантаження, що накладається на Keycloak, що наочно демонструє його роль як потенційного вузького місця у системній архітектурі, особливо під час періодів не особливо високої навантаженості на інфраструктуру. Спостереження за системою під час пікових навантажень вказує на те, що кількість звернень до Keycloak може зростати кратно.

Тому, критично важливим є розробка та впровадження механізму, який дозволить зменшити кількість звернень до Keycloak із сайдкар-компонентів. Один із можливих підходів – є використання кешування токенів, що дозволить знизити частоту безпосередніх запитів до авторизаційного сервісу.

У даній роботі використовується локальний кеш на рівні кожного сайдкара для зберігання токенів, які часто запитуються, що, у свою чергу, мінімізує необхідність здійснення зовнішніх HTTP-запитів при перевірці авторизації.

Діаграма компонентів такого метода надана на рис. 8 та включає наступні кроки:

1. Користувач ініціює запит до мікросервісу 1. При цьому, запит перехоплюється відповідним сайдкаром.

2. Сайдкар 1 аналізує, чи містить локальний кеш значення вхідного токену. У випадку знаходження відповідного токену у локальному кеші, валідація вважається успішною, і звернення до служби Keycloak може бути повністю уникнуте.

3. За умови, якщо локальний кеш не містить вказаного токена, сайдкар ініціює HTTP-запит до API Keycloak для проведення валідації токена.

4. Після перевірки токена сайдкар 1 здійснює виклик іншого модуля 2 для реалізації відповідної бізнес-логіки. Такий процес перехоплюється сайдкаром 2, який повторює логіку, подібну до застосованої сайдкаром 1. Важливо підкреслити, що кожен сайдкар має власний локальний кеш. Тому, присутність токена у кеші одного сайдкара не гарантує його наявність у кеші іншого сайдкара.

Зазначені кроки інтеграції вимагають особливої уваги до деталей реалізації, адже некоректне управління кешем може створити потенційні вразливості,

які дозволять зловмисникам обходити процедуру валідації кешу. Одним з аспектів безпеки, що вимагає уваги, є установка часу життя (time to live, TTL) для записів у кеші. Правильне налаштування TTL є критично важливим для інвалідації застарілих або недійсних токенів. Також, важливо забезпечити вилучення токенів з кешу при видалені користувачів, щоб запобігти несанкціонованому доступу до системи з використанням старих токенів.

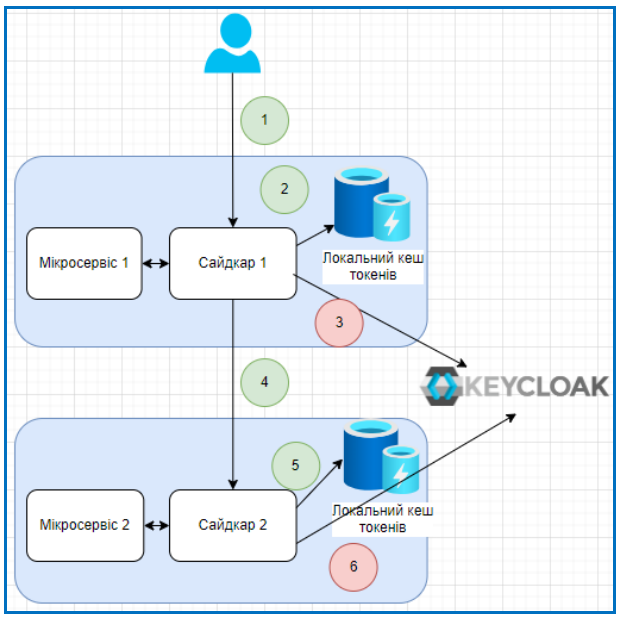

Рис. 8. Діаграма кешування авторизаційних токенів

У процесі реалізації локального кешування токенів важливо визначити необхідний розмір об’єктів, які будуть зберігатися у пам’яті. Розмір токена у Keycloak може варіюватися у залежності від кількості клеймів (claims) та інших атрибутів, які включені до токена. Зазвичай, JWT (JSON Web Tokens), які використовує Keycloak для авторизаційних токенів, мають розмір від кількох сотень байтів до кількох кілобайтів.

Припустимо, що середній розмір токена становить приблизно 2 КБ (2048 байтів). При цьому, розмір локального кешу для 100 токенів буде складати 200 кб. Цей показник є порівняно невеликим, що свідчить про мінімальний вплив на загальні рекомендації з виділення пам’яті для сайдкара.

Слід зазначити, що локальні кеші також знижують ризики збоїв системи, що можуть виникнути у випадку виходу з ладу централізованих кеш-систем, оскільки сайдкару не має необхідності звертатися до сторонніх систем кешування. Порівнюючи представлене рішення з централізованими системами кешування такими як Reddis або Memcached, локальний кеш простіший для реалізації, оскільки не потребує додаткових компонентів, які потім треба ще додатково підтримувати.

Також, використання ізольованих кешів у рамках кожного сайдкара зменшує шанси на несанкціонований доступ до даних з інших частин системи, таких як новий компонент з розподіленим кешом, який містить чутливі дані.

Таким чином, реалізація локального кешу у кожному окремому екземплярі сайдкара без необхідності їх синхронізації представляє собою ефективне та доцільне рішення у високопродуктивній реалізації сайдкара.

Щодо вибору бібліотеки локального кешу слід звернути увагу на Caffeine. Вона оптимізована для високої продуктивності та мінімальних затримок доступу до кешованих даних. Функціонал Caffeine включає можливість управління кешем з використанням алгоритмів LRU (Least Recently Used), які виконуються ефективно завдяки оптимізаціям на рівні компілятора JVM. Ефективне управління пам'яттю та асинхронне API дозволяють уникнути блокувань і конкуренції між потоками, що мінімізує вплив кешу на загальну чутливість системи до затримок.

Згідно з дослідженням [9] Caffeine демонструє значні переваги у швидкості вставлення даних порівняно з іншими популярними бібліотеками кешування, такими як Guava або ConcurrentHashMap.

На рис. 9 показано порівняльний графік запису у кеш між Caffeine, ConcurrentHashMap та Guava.

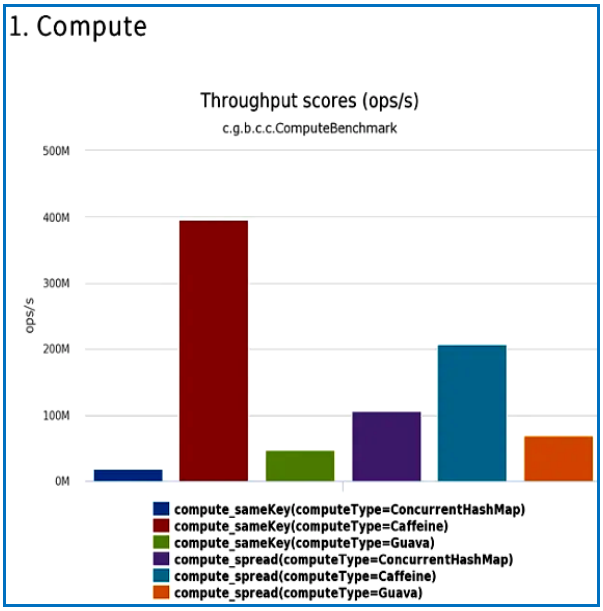

Рис. 9. Переваги Caffeine при запису у кеш

Інтеграція локального кешу Caffeine у сайдкар для зберігання токенів авторизації викликає значне підвищення загальної продуктивності системи. Використання Caffeine кешу дозволяє суттєво зменшити затримки, що пов'язані з постійними зверненнями до сервера автентифікації таких як Keycloak та суттєво зменшити навантаження на них, забезпечуючи відчутне підвищення швидкодії при обробці запитів.

Для перевірки продуктивності реалізованого рішення використано Gatling фреймвок [10]. Проведено оцінку продуктивності під різними видами навантажень:

- вхідні запити з JWT токеном користувача;
- вхідні запити з системним JWT токеном;
- неавторизовані вихідні запити.

Кожен тест можливо налаштувати, використовуючи профіль у файлі .conf.:

```
ingress-10u-300s-50s {
  baseUrl = "http://localhost:19021"
  tracingEnabled = true
  rampUpUsers = 10
  rampUpDuration = 1s
  testDuration = 300s
  tokenRefreshInterval = 60s
  config {
    type = "ingress"
    includeUserJwt = true
    includeSystemJwt = true
    requestsFile = "igress-requests.jsonl"
  }}.
```

Для тесту сайдкар був максимально ізольованим від інших системних служб. Усі запити на зовнішні служби та сервіси були імітованими за допомогою інструменту Wiremock. Це необхідно, щоб точно оцінити саме продуктивність окремого компоненту, сайдкара у нашому випадку, від оцінки всієї системи в цілому. Для даного тесту Keycloak містив тенант з 100 користувачами та 10 випадковими ролями.

Ресурси сайдкар компонента для різних тестів виділялися у таких діапазонах:

CPU Reservation: 128 / 256 / 512 / 1024 (0.125 / 0.25 / 0.5 / 1.0 vCPU);

CPU Limit: 128 / 256 / 512 / 1024 (0.125 / 0.25 / 0.5 / 1.0 vCPU);

Memory Reservation: 192 MB;

Memory Limit: 224 MB.

Для мікросервісу, з яким пов'язаний сайдкар:

CPU limit: 1024 (1.0 vCPU);

Memory Limit: 400 MB;

Response Delay: 50 / 200 / 1000 ms;

Request Body: (jsonObject with key: 10 random chars, value: 100 random chars), number of keys = 50 / 500 / 5000.

Для сервера авторизації Keycloak:

CPU Limit: 3072 (3.0 vCPU);

CPU Reservation: 2048 (2.0 vCPU);

Memory Limit: 1200 MB;

Memory Reservation: 1200 MB.

Тест проводився протягом 600 секунд з використанням 100 віртуальних користувачів (V_USERS). У якості параметру RAMP_UP використовувалося значення у 550 секунд. Це означає, що V_USERS будуть поступово додаватися до тесту протягом 9 хв. та 10 с.

У даному випадку крива навантаження буде рівномірно зростати, доки не буде досягнуто повного об'єму одночасних користувачів. Таке плавне збільшення навантаження може допомогти виявити потенційні проблеми зі стійкістю та продуктивністю системи при поступовому зростанні користувачів, як представлено на рис. 10.

Параметри першого тесту Gatling, виконаного у рамках дослідження, включали затримку у 50 мілісекунд (ms), виділення обчислювальних ресурсів на рівні 0,1 віртуального центрального процесора (vCPU), налаштування одного обробника подій (Event Loop) та використання п'яти робочих потоків (Thread Workers). Такі параметри були обрані для тестування впливу конфігурації сервісу на його здатність обробляти навантаження та забезпечувати стабільну роботу в умовах різного рівня запитів.

У табл. 1 представлено результати вхідних викликів, які оброблялися сайдкаром для проведеного тесту 1.

Загальна кількість запитів склала 27,713, з яких усі були успішно оброблені без помилок (0% помилок). Середня інтенсивність запитів становила 46,19 запитів за секунду.

Мінімальний час на відгук склав 19 мілісекунд. Медіанний (50-й перцентиль) час відгуку становить 5360 мс, а час відгуку для 75-го перцентиля – 8230 мс. Час відгуку для 95-го перцентиля дорівнював 11596 мс, а для 99-го перцентиля – 12309 мс.

Максимальний зафіксований час на відгук склав 13851 мс, середній час відгуку – 5779 мс, зі стандартним відхиленням 3293 мс.

Отримані результати свідчать про високу продуктивність та стабільність сайдкару при обробці HTTP запитів.

Для наступного тесту 2 були збільшені процесорні потужності для сайдкару з 0,1 vCPU до 0,5 vCPU (табл. 2).

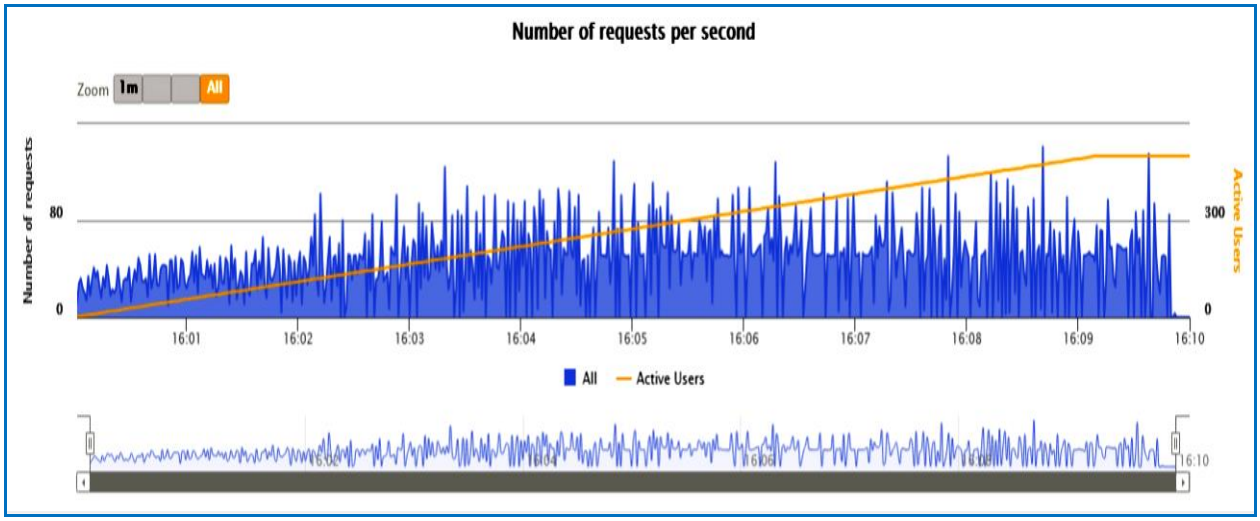

Рис. 10. Графік збільшення кількості одночасний запитів під час тесту

Таблиця 1 – **Тест №1 з 0,1 vCPU**

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
All Requests	27713	27713	0	0	46,19	19	5360	8230	11596	12309	13851	5779	3293
Authorize user	1141	1141	0	0	1,9	19	25	39	70	123	164	34	20
GET request	6096	6096	0	0	10,16	63	5611	8511	11605	12241	13354	6072	3149
POST request	4821	4821	0	0	8,04	56	5577	8385	11580	12204	13305	6013	3139
DELETE request	10703	10703	0	0	17,84	55	5534	8436	11508	12165	13851	6004	3132
PUT request	4952	4952	0	0	8,25	55	5616	8335	11607	12156	13305	6030	3123

Таблиця 2 – **Тест №2 з 0,5 vCPU**

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
All Requests	176426	176426	0	0	294,04	19	707	1407	2292	2608	3411	920	704
Authorize user	1291	1291	0	0	2,15	19	31	48	110	185	271	44	32
GET request	40164	40164	0	0	66,94	52	713	1403	2310	2618	3313	927	702
POST request	31714	31714	0	0	52,86	53	706	1408	2288	2584	3190	924	705
DELETE request	70577	70577	0	0	117,63	52	700	1402	2286	2587	3411	925	702
PUT request	32680	32680	0	0	54,47	52	715	1405	2229	2580	3206	930	701

Після аналізу отриманих результатів другого Gatling тесту сайдкару з конфігурацією 0,5 vCPU, можливо стверджувати, що збільшення процесорних ресурсів має значний вплив на продуктивність сайдкару.

Істотні зміни спостерігаються у наступних параметрах:

1. Загальна кількість запитів за той самий час значно зросла з 27,713 до 176,426.
2. Інтенсивність запитів збільшена з 46,19 до 294,04 запитів за секунду.
3. Середній час на відповіді значно покращився з 5779 мс до 920 мс.
4. Мінімальний, медіанний (50-й перцентиль), 75-й перцентиль, 95-й перцентиль та 99-й перцентиль часу на відповіді значно покращились, вказуючи на швидшу обробку запитів.

Дані зміни демонструють, що сайдкар збільшив свою продуктивність та здатність обробляти більшу кількість запитів за одиницю часу при збільшенні віртуальних центральних процесорів. Таке свідчить про те, що сайдкар ефективно використовує додаткові процесорні ресурси. Це стає ключовим аспектом для оптимізації продуктивності у виробничих середовищах, де вимоги до швидкодії та масштабованості постійно зростають.

За результатами аналізу другого Gatling тесту сайдкару з конфігурацією 0,5 віртуального центрального процесора (vCPU), можливо стверджувати про наступні ключові пікові значення, які вказують на оптимальну продуктивність:

- максимальна кількість запитів за секунду: загальна інтенсивність запитів досягла 294,04 запитів за секунду. Таке значення вказує на те, наскільки ефективно сайдкар може обробляти велику кількість вхідних запитів;
- мінімальний час на відповіді зафіксовано для авторизації користувача – 19 ms.

Висновки

Таким чином, використання open-source рішення Keycloak для делегації авторизації у сайдкарі є ефективним рішенням як з погляду безпеки, так і з точки зору оптимізації продуктивності.

Централізація авторизації у сайдкарі дозволила зменшити навантаження на мікросервіси, звільняючи їх від необхідності обробляти авторизаційну логіку та дозволяючи сфокусуватися на виконанні бізнес-логіки.

Також, розміщення авторизації у сайдкарі сприяє консолідації безпеки, оскільки уся інформація про авторизацію та її логіка управління концент-

руються у одному місці. Використання Keycloak дає змогу легко інтегрувати сторонні постачальники ідентичності через стандартизовані протоколи, такі як OpenID Connect або SAML. Це надає більшу гнучкість та розширює можливості щодо забезпечення безпеки без значних зусиль з розробки на стороні мікросервісів.

Завдяки комплексному підходу щодо оптимізації у конфігурацію та архітектуру сайдкару, можливо досягти значного підвищення продуктивності. Отримані дані тестувань Gatling підтвердили результати.

Одним з ключових внесків у оптимізацію є рішення про кешування токенів авторизації Keycloak.

Використання бібліотеки Caffeine для таких цілей дозволяє значно скоротити затримки, що пов'язані з частими запитами на авторизацію.

За результатами впровадження розроблених пропозицій з боку оптимізації встановлено, що сайдкар не тільки значно покращує свою продуктивність, але і демонструє здатність щодо ефективного масштабування для задоволення зростаючих вимог сучасних веб-додатків та сервісів.

СПИСОК ЛІТЕРАТУРИ

1. K. Salah, R. N. Calheiros, and R. Buyya, "Security challenges in microservice architectures: A comprehensive survey," IEEE Trans. Cloud Comput., vol. 9, no. 3, pp. 1185-1206, Jul. 2019.
2. Документація Keycloak: https://www.keycloak.org/documentation.html
3. S. Nkomo, "Managing API credentials for microservices security," J. Network and Systems Management, vol. 28, no. 2, pp. 345-367, Apr. 2020.
4. K. Ueda, T. Fujibayashi, and H. Suzuki, "Role-Based Access Control in Microservice Architectures," Journal of Information Security and Applications, vol. 56, pp. 102-114, Mar. 2021.
5. D. Wang, H. Jiang, and L. Meng, "Secure Authorization Mechanism Using Sidecar Architecture in Microservices," in Proceedings of the Symposium on Applied Computing, pp. 1522-1529, March 2021.
6. Y. Zhang, P. Li, and J. Xu, "Enhancing Microservices Security via Sidecar Proxies: A Case Study with Envoy," IEEE Access, vol. 10, pp. 23452-23463, 2022.
7. С. С. Бульба, О. В. Коломійцев, О. І. Соловйова, С. В. Носко. Засоби побудови додаткового рівня системи комунікацій у мікро-сервісній архітектурі. Грааль науки : міжнар. наук. журнал. – Вінниця : ГО «Європейська наукова платформа»;НУ «Інститут науково-технічної інтеграції та співпраці», 2024. – No 46. – 651-659 с. DOI 10.36074/grail-of-science.
8. Kuchuk, N., Shiman, A., Filonenko, A. and Bulba, S. 2021. Розрахунок ефективності використання обчислювальних ресурсів самовідновлювальної комп'ютерної системи. Системи управління, навігації та зв'язку. Збірник наукових праць. 3, 65 (Вер 2021), 92-95.
9. DevOps блог: "Caffeine Cache". URL: https://blog.devops.dev/easy-to-use-caffeine-cache-1-3db5861f6f39
10. Gatling фреймворк: веб-сайт. URL: https://gatling.io/

Received (Надійшла) 29.11.2024
Accepted for publication (Прийнята до друку) 19.02.2025

Suggestions for authorization in the sidekick component of microservice architecture

S. Nosko, S. Bulba, O. Kolomiitsev, D. Lysytsia, H. Molchanov

Abstract. The article develops proposals on the feasibility of implementing authorization in the sidekick component of a microservice, which allows separating business logic from authorization tasks, logging, caching, etc. Such an implementation allows the main business modules to remain focused exclusively on their business logic, changing only when there are updates to business procedures, and tasks that are common to almost any modern solution to be transferred to auxiliary components such as a sidekick. Proper implementation of authorization is a key aspect of any system, it is very important and requires taking into account a large number of features and using best practices to ensure the safety of data from unauthorized access. The possibilities of the open-source Keycloak solution are considered. This solution is a very popular service for authentication/authorization, it supports standard identification protocols, such as: OpenID Connect, OAuth 2.0, SAML 2.0. The solution supports multi-factor authentication, supports containerization, and can be easily distributed in a microservices environment like Kubernates and integrated with various external services, including Google, Facebook, etc. The architecture and proposals necessary for integration with the sidekick component are developed. A study on the performance of the developed solution was conducted. Numerical values were obtained and presented in tables and graphs. Additionally, it can integrate with various external services, including Google, Facebook, and others. The article presents the architecture and method required for integration with a sidecar component and includes a performance analysis of the proposed solution. To enhance the performance further, the sidecar architecture incorporates a local caching mechanism, utilizing Caffeine cache — known for its high performance and effectiveness. This cache stores authorization tokens, significantly decreasing the volume of calls required to Keycloak by reusing the tokens stored locally until they expire. This mechanism reduces reliance on central Keycloak servers and minimizes latency, providing a performance boost through decreased network traffic and accelerated token retrieval processes. Extensive performance testing was conducted using the Gatling framework to validate the integration and caching strategy. These tests demonstrated that the sidecar configuration, equipped with the local Caffeine cache, maintained consistent performance under varying loads and was capable of horizontal scaling without degradation in response times. Moreover, the reduced load on the Keycloak servers showcased the effectiveness of the caching approach in minimizing backend calls for token validations. The approach confirms the robustness and scalability of the sidecar, poised to handle increased loads efficiently while safeguarding sensitive authorization data.

Keywords: Sidecar, microservices, authorization, Keycloak, performance, cache, containers, Kubernetes, scalability, open-source, Gatling, resource allocation, load balancing, web server, information system.