УДК 004.42
DOI: 10.31673/2412-9070.2020.023942

О. Б. ПРИДИБАЙЛО, ст. викладач, здобувач;
Р. В. ПРИДИБАЙЛО, провідний інженер,
Державний університет телекомунікацій, Київ

МІКРОСЕРВІСНИЙ ПІДХІД ДО РОЗПОДІЛЕНИХ АРХІТЕКТУР ДОДАТКІВ

Розглянуто керування трафіком, яке відбувається за допомогою сервісної сітки. Сервісна сітка вигідна тим, що вирішує складності, які виникають в додатках, заснованих на наборі невеликих сервісів, кожний з яких функціонує у власному процесі і комунікує з іншими механізмами, як правило HTTP, інакше кажучи, працює з мікросервісами. Розглядуваній технології притаманні великі переваги, за допомогою яких є можливість під'єднувати мережі мікросервісів, а також керувати ними, забезпечуючи їхню безпеку незалежно від середовища виконання, джерела і розробника. Сервісна сітка дає змогу керувати вхідним та вихідним обсягами інформації у такий спосіб, щоб була змога відслідковувати час очікування відгуку на запит, повторні спроби запитів та балансування навантаження на комп'ютерні системи; забезпечує спостереження за процесом покрокового виконання програми, за системою постійного стеження, а також забезпечує безпеку користувача. Дуже важливим є те, що сервісна сітка функціонує в мережі і реалізує подання інформації про систему користувача та процес збору, агрегації та аналізу цих даних для вдосконалення характеристик і поведінки компонентів системи. Для коректної роботи сервісної сітки існує програмна частина архітектури додатку, що забезпечує безпечну, швидку та надійну взаємодію між дискретними програмними компонентами — сервісами. Запропоновано кілька схем, які наочно показують архітектуру сервісної сітки і загальну архітектуру додатку, який можна реально створити, перевірити та запустити на виконання на хмарних платформах. Запуск сервісної сітки відбувається на платформах Google, що є дуже зручним для використання, оскільки не потребує додаткових затрат і має дуже простий код для підімкнення. За допомогою service mesh можна контролювати потоки трафіку і виклики API між службами, а також отримувати огляд певного трафіку. Це робить запити та мережу більш надійними навіть за несприятливих умов, водночас даючи змогу користувачам виявляти проблеми, перш ніж вони стануть несприятливими умовами для розробки та використання додатків.

Ключові слова: сервісна сітка; мікросервіси; керування трафіком; панель даних; панель керування; маршрутизація трафіку; віртуальні сервіси; мікросервісні сітки; мережа мікросервісів; додатки; середовище налаштування; шифрування; автентифікація; метрики; моніторинг; мікросервісна архітектура; проксі.

Вступ

Розроблення транзакційних серверних додатків, пов'язаних із інтернетом, приймання запитів із зовнішнього світу і відповіді на них протягом короткого часу (наприклад, веб-додаток, API-сервер) та й переважна більшість інших сучасних додатків, якщо вони реалізуються як набір із сервісів, котрі синхронно взаємодіють один з одним, — це все створення сучасного серверного програмного забезпечення. Розвиток хмарних технологій дає можливість використовувати мікросервіси для розроблення архітектур додатків. Вони мають бути надійними, безпечними, а розробник повинен мати можливість спостерігати за тим, як дані додатки працюють. Усі ці завдання є критично важливими для стабільної роботи, тому постала потреба оптимізації зв'язків між службами об'єктно-орієнтованих додатків. Розв'язання таких завдань покладено на сервісну сітку (*service mesh*). ***Service mesh*** — це виокремлений шар інфраструктури для забезпечення безпечної, швидкої і надійної взаємодії між сервісами. Якщо створюється додаток для запуску в хмарі (тобто *cloud native*), потрібна сервісна сітка.

Мікросервіси розробники використовують для налаштування перенесення та розгортання, підімкнення, захисту, контролю та спостереження за сервісами. Сервісна сітка має повністю відкритий код, вона прозоро розміщується на наявних розподілених додатках.

Це також платформа, зокрема API-інтерфейси, які уможливлюють інтегрування в будь-яку платформу ведення журналів, телеметрію або систему телекомунікацій. Різноманітний набір функцій технологій розгортання додатків дає змогу успішно й ефективно запускати архітектуру розподілених мікросервісів і забезпечує єдиний спосіб захисту, підімкнення та моніторингу мікросервісів. Оптимально використовувати service mesh за потреби надання мінісервісів і мікросервісних операцій.

Сітка обслуговування — це рівень інфраструктури, який дозволяє керувати зв'язком між мікросервісами певного застосування. У міру того як зростає кількість розробників, що працюють із мікросервісами, сервісні сітки розвиваються, аби спростити цю роботу і підвищити її ефективність завдяки об'єднанню спільних завдань керування і адміністрування в розподіленій установці.

Застосування мікросервісного підходу до архітектури додатків включає в себе розбиття програми на набір слабкозв'язаних сервісів. Такий підхід дає певні переваги: команди можуть швидко виконувати ітерації і масштабувати, використовуючи більш широкий спектр інструментів і мов [1].

Основна частина

Сервісна сітка (*service mesh*, або мережа мікросервісів) є розподіленим програмним забезпеченням проміжного шару (middleware), що оптимізує

зв'язок між службами додатків. Вона забезпечує спрощене посередництво для зв'язку між службами і підтримує такі функції, як автентифікація, авторизація, шифрування, виявлення сервісів, маршрутизація запитів, балансування навантаження, самовідновлення та інструментарій послуг.

Водночас мікросервіси висувають нові завдання через складність операцій, узгодженості даних і безпеки. Сервісні сітки призначено для вирішення деяких із цих проблем, пропонуючи детальний рівень контролю над тим, як сервіси взаємодіють один із одним. Зокрема, вони пропонують розробникам такі функції.

1. ***Виявлення сервісів.*** У розподіленому середовищі необхідно знати, як під'єднуватися до сервісів та чи є вони доступними. Розташування примірників сервісу в мережі призначається динамічно, і інформація про них постійно змінюється в міру створення і знищення контейнерів через автоматичне масштабування, оновлення та збої. Тому було запропоновано підхід виявлення сервісів на основі DNS за замовчуванням. За його допомогою можна шукати сервіси та сервісні порти, а також виконувати зворотне перетворення IP, використовуючи загальні угоди іменування DNS. У загальному випадку запис має такий вигляд:

service.namespace.svc.cluster.local

2. ***Маршрутизація та настроювання трафіку.*** Керування трафіком у розподіленому середовищі означає керування тим, як трафік потрапляє в кластер користувача і як він спрямовується на його сервіси. Чим більше контролю і специфіки має користувач під час налагодження зовнішнього і внутрішнього трафіку, тим досконаліше він зможе налаштувати дані процеси. Ключовим моментом для успішного проведення операцій часткового запуску сервісу в експлуатацію, перенесення додатків на нові версії або тестування певних сервісів за допомогою внесення несправностей є можливість визначити, який обсяг трафіку отримують сервіси та звідки він надходить.

3. ***Шифрування і автентифікація (авторизація):***

• *шифрування* може належати до з'єднань між кінцевими користувачами і сервісами, секретних даних, кінцевих точок у площині керування зв'язком між компонентами робочого кластера і головними компонентами;

• *автентифікація:* запити API прив'язані до облікових записів користувачів або сервісів, які мають розпізнаватися. Існує кілька різних способів керування обліковими даними: статичні маркери, маркери початкового завантаження, клієнтські сертифікати і зовнішні інструменти, такі як OpenID Connect;

• *авторизація:* є різні модулі авторизації, які дають можливість визначати доступ на основі ролей, атрибутів і інших спеціалізованих функцій. Оскільки всі запити до сервера API за замовчуванням відхиляються, кожна частина запиту API має визначатися політикою авторизації.

4. ***Метрики і моніторинг.*** Розподілені середовища змінили вимоги до метрик і моніторингу. Інструменти моніторингу мають бути адаптивні, зважати на часті зміни в сервісах і мережних адресах та давати змогу враховувати обсяг і тип інформації, що передається між сервісами. У сервісній сітці Istio структура конвеєра повної метрики є частиною дизайну. Супровідники Envoy, що працюють на рівні подів, передають метрики в Mixer, який керує політиками і телеметрією. За допомогою цих метрик і інструментів візуалізації можна отримати централізований доступ до поточної інформації про сервіси і робочі навантаження [2].

Сервісні сітки можуть спростити процес роботи із загальними компонентами в мікросервісній архітектурі, у деяких випадках навіть розширюючи функціональність цих компонентів.

Функції, що надаються сервісною сіткою, не просто критично важливі. Вони застосовуються до всіх сервісів у додатку незалежно від того, якою мовою і хто їх написав, який фреймворк використовують, як їх було розгорнуто і від усіх інших тонкощів їх розроблення і застосування. Мережа мікросервісів не тільки надає єдині функціональні можливості для всього стека — вона здійснює це у такий спосіб, який не потребує редагування програми. Фундаментальна основа функціональності сервісної сітки, зокрема завдання з налаштування, оновлення, експлуатації, обслуговування тощо, охоплює виключно рівень платформи і не залежить від додатка. Додаток може змінюватися, не зачіпаючи сервісну сітку. У свою чергу, вона може змінюватися без будь-якої участі додатка.

Концепція service mesh як окремого шару пов'язана зі збільшенням кількості додатків, створюваних спеціально для хмарних оточень. У такій хмарній моделі єдиний додаток може складатися із сотень сервісів, у кожного сервісу можуть бути тисячі примірників, і у кожного примірника може змінюватись стан залежно від динамічного планування. Тому взаємодія сервісів є фундаментальною частиною поведінки середовища виконання, а керування ним дуже важливо для підтримання продуктивності і надійності.

Розглянемо структуру service mesh. Це є певна кількість віддалених комп'ютерів-посередників (проксі-серверів) і набір керуючих процесів. Проксі в сукупності дістали назву *площина даних* (data plane), а керуючі процеси йменуються *площиною керування* (control plane) (рис. 1) [3].

Площина керування складається з набору компонентів, які забезпечують усю механіку, необхідну площині даних, щоб працювати скоординовано,

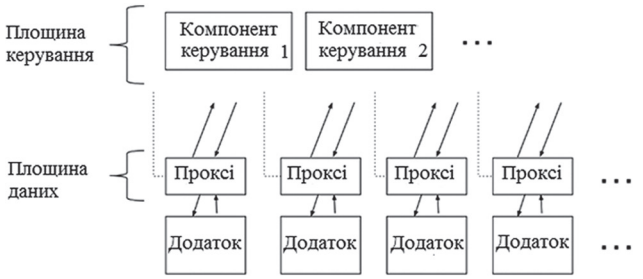

Рис. 1. Загальна схема архітектури сервісної сітки (service mesh)

зокрема виявлення сервісів, випуск сертифікатів TLS, агрегацію метрик тощо. Площина даних інформує площину керування про процеси, які в ній відбуваються, тоді як площина керування надає API, що дає можливість змінювати і відстежувати процеси площини даних як єдиного цілого. Control plane включає в себе кілька різних компонентів, які збирають метрики з проксі-серверів, а також компоненти для виявлення сервісів та центр сертифікації. Площина керування координує поведінку проксі і забезпечує доступ до оператора, до API, уможливлюючи маніпулювання мережею і вимірювання її як єдине ціле.

Площина даних перехоплює виклики між сервісами та працює з різноманітними додатками. Проксі-сервери спрямовують виклики до сервісів і від них (строго кажучи, вони виконують функцію проксі і зворотних проксі, обробляючи як вхідні, так і вихідні дзвінки). Також вони реалізують набір функцій, концентруючись на виклики між сервісами.

Як приклад наведемо схему площини керування та площини даних у Linkerd (рис. 2) [4].

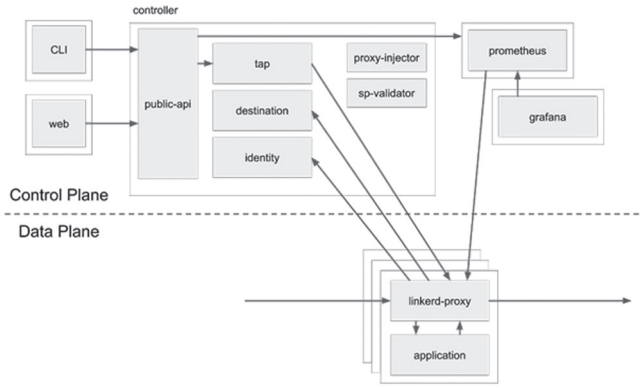

Рис. 2. Схема sidecar-контейнер

Архітектура service mesh має кілька важливих наслідків. По-перше, оскільки завдання проксі — перехоплювати виклики між сервісами, сервісна сітка має сенс тільки в тому разі, якщо розроблений додаток було створено як набір сервісів. Сітку можна використовувати з монолітами, але це явно надлишково заради одного єдиного проксі, та й її функціонал навряд чи буде затребувано.

По-друге, сервісна сітка потребує величезної кількості проксі. Настільки активне використання проксі саме по собі призводить до низки додаткових ускладнень:

♦ проксі в data plane мають бути швидкими, оскільки на кожний виклик припадає пара звернень до проксі: одне — з боку клієнта, одне — з боку сервера;

♦ проксі мають бути невеликими і легкими — кожний споживатиме ресурси пам'яті і CPU, і це споживання буде лінійно зростати разом із додатком;

♦ знадобиться механізм для розгортання та оновлення великої кількості проксі.

Але, незважаючи на певні ускладнення, операційні витрати, пов'язані з розгортанням цих проксі, можуть бути значно знижені завдяки деяким змінам, що відбуваються в екосистемі. Подібний пристрій — це насправді відмінний спосіб ввести додаткову логіку в систему. І не тільки тому, що за допомогою сервісної сітки можна додати безліч нових функцій, а й тому, що це можна зробити, не втручаючись в екосистему. Уся модель сервісної сітки ґрунтується на такому постулаті: у мультисервісній системі, незалежно від того, що роблять окремі сервіси, трафік між ними є ідеальною точкою для додавання функціональності [5].

Висновки

Отже, даний підхід вирішує проблему оптимізації використання апаратного та програмного забезпечення завдяки високій продуктивності та можливості зберігання інформації на хмарних платформах замість жорстких дисків на комп'ютерах та серверах. Віртуальні сервіси, що працюють за певними правилами, використовують ключові елементи компонентів функціональності маршрутизації трафіку сервісних сіток. Для налаштовування сервісів у сітці сервісу використовують віртуальний сервіс, який дає змогу користувачеві, ґрунтуючись на базових можливостях підімкнення та обертання, розгортати багато userspace-проксі. Кожний віртуальний сервіс є наявним із набору правил маршрутизації, які оцінюються за дотриманням, отже, сервісна сітка дозволяє встановити зв'язок кожного даного запиту до віртуальної служби з конкретним реальним пунктом призначення в сітці. Така структура може потребувати кількох віртуальних сервісів або жодного залежно від конкретного варіанта використання.

Найбільша перевага застосування service mesh полягає в тому, що дана технологія знижує навантаження на розробників, що може підвищити їх продуктивність і допомогти у швидшому доставлянні додатків. У більш довгостроковій перспективі, на думку експертів, використання сервісної сітки здатне допомогти компаніям гарантувати однакове застосування певних стандартів і політик

у різних додатках. Це пов'язано з тим, що service mesh забезпечує керування трафіком, що дає низку додаткових переваг у мікросервісних середовищах, зокрема доступність, відмовостійкість, динамічність, масштабованість і безпеку.

Список використаної літератури

*1. **Никульчев Е. В., Паяин С. В., Плужник Е. В.** Динамическое управление трафиком программно-конфигурируемых сетей в облачной инфраструктуре // Вестник РГРТУ. 2013. № 3.*

*2. **Анализ** моделей управления трафиком в сетях АСУП на основе технологии MPLS / В. Т. Еременко, С. В. Еременко, Д. В. Анисимов [и др.] // Информационные системы и технологии. 2013. №1.*

*3. **Алиев Т. И., Муравьева-Витковская Л. А.** Приоритетные стратегии управления трафиком в мультисервисных компьютерных сетях // Известия высш. учеб. заведений. Приборостроение. 2011. Т. 54. №6.*

*4. **Контроль,** измерение и интеллектуальное управление трафиком / А. А. Алейников, К. З. Билятдинов, А. В. Красов, М. В. Левин // Центр науч.-информ. технологий «Астерион». СПбб 2016. 92 с.*

*5. **Exploring** and troubleshooting istio issues / T. Lange, A. Shribman, E. Raichstein, K Barabash // Publication: SYSTOR'19: Proceedings of the 12th ACM International Conference on Systems and Storage. 2019. С. 196.*

*6. **Sharma R., Singh A.** Getting Started with Istio Service Mesh // Apress, Berkeley, CA. 2020. 321 с.*

О. Б. Придыбайло, Р. В. Придыбайло

МИКРОСЕРВИСНЫЙ ПОДХОД К РАСПРЕДЕЛЕННОЙ АРХИТЕКТУРЕ ПРИЛОЖЕНИЙ

Рассмотрено, как с помощью сервисной сетки происходит управление трафиком. Сервисная сетка выгодна тем, что решает сложности, которые возникают в приложениях, основанных на наборе небольших сервисов, каждый из которых работает в своем процессе и коммуницирует с другими механизмами, как правило HTTP, иначе говоря, работает с микросервисами. Рассматриваемой технологии присущи большие преимущества, с помощью которых существует возможность подключать сети микросервисов, а также управлять ими, обеспечивая их безопасность независимо от среды выполнения, источника и разработчика. Сервисная сетка позволяет управлять входным и выходным объемами информации таким образом, чтобы была возможность отслеживать время ожидания отклика на запрос, повторные попытки запросов и балансировки нагрузки на компьютерные системы; обеспечивает наблюдение за процессом пошагового выполнения программы по системе постоянного слежения, а также обеспечивает безопасность пользователя. Очень важно, что сервисная сетка функционирует в сети и реализует представление информации о системе пользователя и процессе сбора, агрегации и анализа этих данных для совершенствования характеристик и поведения компонентов системы. Для корректной работы сервисной сетки существует программная часть архитектуры приложения, что обеспечивает безопасное, быстрое и надежное взаимодействие между дискретными программными компонентами — сервисами. Предложено несколько схем, которые наглядно показывают архитектуру сервисной сетки и общую архитектуру приложения, которое можно реально создать, проверить и запустить на выполнение на облачных платформах. Запуск сервисной происходит на платформах Google, что очень удобно для использования, поскольку не требует дополнительных затрат и имеет очень простой код для подключения. С помощью service mesh можно контролировать потоки трафика и вызовы API между службами, а также получать обзор определенного трафика. Это делает запросы и сеть более надежными даже в неблагоприятных условиях, одновременно позволяя пользователям выявлять проблемы, прежде чем они станут неблагоприятными условиями для разработки и использования приложений.

Ключевые слова: сервисная сетка; микросервисы; управления трафиком; панель данных; панель управления; маршрутизация трафика; виртуальные сервисы; микросервисные сетки; сеть микросервисов; приложения; среда настройки; шифрование; аутентификация; метрики; мониторинг; микросервисна архитектура; прокси.

O. B. Prydybailo, R. V. Prydybailo

MICRO-SERVICE APPROACH TO THE DISTRIBUTED ARCHITECTURE OF APPLICATIONS

The article considers how traffic is managed with the help of a service grid. The service grid is advantageous in that it solves the difficulties that arise in applications based on a set of small services, each of which works in its own process and communicates with other mechanisms, usually HTTP, in other words, works with microservices. In the technology discussed in this article, there are great advantages that allow you to connect networks of microservices, as well as manage them, ensure their security regardless of the runtime, source and developer. The service network allows you to control the input and output of information so that you can track the waiting time for a response to a request, retry requests and load balancing on computer systems; provides monitoring of the process of step-by-step execution of the program, the system of constant tracking, as well as ensures the security of the user. It is very important that the service network operates in the network and implements the presentation of information about user systems and the process of collecting, aggregating and analyzing this data to improve the characteristics and behavior of system components. For the correct operation of the service network, there is a software part of the application architecture that provides safe, fast and reliable interaction between discrete software components - services. Also, in this article there are several diagrams that clearly show the architecture of the service network and the general architecture of the application, which can be actually created, tested and run on cloud platforms. The service is launched on Google platforms, which is very convenient to use, because it does not require additional costs and has a very simple code to connect. With service mesh, you can monitor traffic flows and API calls between services, as well as get an overview of specific traffic. This makes queries and the network more reliable, even in adverse conditions, and at the same time allows users to detect problems before they become unfavorable for the development and use of applications.

Keywords: service grid; microservices; traffic management; data panel; control panel; traffic routing; virtual services; microservice grids; microservice network; applications; configuration environment; encryption; authentication; metrics; monitoring; microservice architecture.