DOI 10.37882/2223–2966.2022.03.16

ПОДХОДЫ К ФОРМИРОВАНИЮ БЕЗОПАСНОСТИ В МИКРОСЕРВИСНОЙ АРХИТЕКТУРЕ

APPROACHES TO THE FORMATION OF SECURITY IN MICROSERVICE ARCHITECTURE

I. Irbitskiy
A. Romanenkov
K. Stulnikov
N. Udalov

Summary. The rapid development and spread of network cloud services by the early 2010s led to disappointment in the classic, so-called monolithic version of the application architecture. Due to the complexity of individual modules, often representing entire software systems, as well as due to the need to ensure compatibility between them through standard protocols, making any changes and additions has become a non-trivial task that takes too much time.

As an answer to this challenge, the architecture of microservices was proposed as a distributed system of the simplest and easily replaceable modules that perform, if possible, a single elementary function. At the same time, the microservice system has a symmetrical, peer-to-peer, rather than hierarchical organization, which eliminates the need for a complex organization of relationships. Services communicate with each other and with clients using lightweight protocols, for example, via HTTP or text messages. As a result, a system is created that is easy to deploy and upgrade with automatic development and update functions.

By 2021, microservice architecture is in the center of attention of specialists and not only: it is written about in blogs, in social networks, discussed in the press and at various conferences. The successful implementation of microservices is announced by representatives of Amazon, Google, Netflix and Twitter. In Russia, the experience of switching to microservices was reported by large banks, as well as, for example, M. Video-Eldorado and MegaFon.

Keywords: microservice, architecture, development, server, high-load software, attack, DDoS.

Ирбитский Илья Сергеевич
Аспирант, Московский авиационный институт (национальный исследовательский университет)
scarletsurge.u@gmail.com
Романенков Александр Михайлович
К.т.н., доцент, Московский авиационный институт (национальный исследовательский университет); С.н.с., ФИЦ ИУ РАН
romanaleks@gmail.com
Стульников Кирилл Тимурович
Московский авиационный институт (национальный исследовательский университет)
frostik0409@gmail.com
Удалов Никита Николаевич
Московский авиационный институт (национальный исследовательский университет)
nnudalov@gmail.com

Аннотация. Стремительное развитие и распространение сетевых облачных сервисов к началу 2010-х годов привело к разочарованию в классическом, так называемом монолитном варианте архитектуры приложений. Из-за сложности отдельных модулей, зачастую представляющих собой целые программные системы, а также из-за необходимости обеспечивать совместимость между ними посредством стандартных протоколов, внесение любых изменений и дополнений стало нетривиальной задачей, отнимающей слишком много времени.

В качестве ответа на этот вызов была предложена архитектура микросервисов как распределенная система простейших и легко заменяемых модулей, выполняющих по возможности единственную элементарную функцию. При этом микросервисная система имеет симметричную, одноранговую, а не иерархическую организацию, что снимает необходимость в сложной организации взаимосвязей. Сервисы связываются между собой и с клиентами с использованием лёгких протоколов, например, через HTTP или при помощи текстовых сообщений. В результате создаётся система, простая в развёртывании и модернизации с функциями автоматической разработки и обновления.

К 2021 году микросервисная архитектура находится в центре внимания специалистов и не только: о ней пишут в блогах, в соцсетях, обсуждают в прессе и на различных конференциях. Об успешном внедрении микросервисов заявляют представители Amazon, Google, Netflix и Twitter. В России об опыте перехода на микросервисы сообщали крупные банки, а также, например, «М. Видео-Эльдорадо» и «МегаФон».

Ключевые слова: микросервис, архитектура, разработка, сервер, высоконагруженное программное обеспечение, атака, DDoS.

Введение

По мере того как все больше организаций переходят от монолитной архитектуры к распределенной архитектуре на основе микросервисов, проблемы безопасности становятся более актуальными. В микросервисной архитектуре возможности для атак значительно увеличиваются из-за различных сетевых подключений и API, используемых для общения между всеми компонентами. При данном подходе логические модули могут быть распределены по сервисам, которые в свою очередь могут быть обернуты в специальные контейнеры и распределены по множеству различных систем/хостов и т.д. Так как они должны функционировать согласованно, ставится задача проектирования правильной архитектуры сети и взаимодействия между ними. Это также означает, что каждый контейнер должен надлежащим образом обслуживаться, управляться и защищаться, на что необходимо чрезвычайно много времени без использования надлежащих технологий.

При использовании традиционного монолитного приложения компоненты и службы обычно размещаются на одном или нескольких серверах в одной сети, что облегчает их взаимодействие. Из-за большого количества открытых API, портов и компонентов традиционные межсетевые экраны не могут обеспечивать должный уровень безопасности для них. Эти проблемы делают развёртывание микросервисов более уязвимым для различных киберугроз, таких как «man-in-middle», инъекционные атаки, межсайтовый скриптинг, DDoS [1]. По сути, это означает, что для запуска проекта, состоящего из нескольких микросервисов, потребуется детальное обеспечение контроля безопасности, которое включает в себя несколько решений.

С точки зрения безопасности контейнеры и микросервисы обладают некоторыми очевидными преимуществами в плане безопасности [2], особенно с учетом того, что компоненты и службы приложений изолированы. Безопасность микросервисов может быть реализована на гораздо более детальном уровне, с элементами управления, применяемыми к конкретным службам, API и сетевым каналам связи.

Микросервисы действительно предоставляют возможность реализовать стратегию углубленной защиты, но способ реализации средств контроля безопасности — это огромный отход от традиционных методов. В архитектуре микросервисов существует множество транзакций и взаимодействий. Таким образом, безопасность [3] компонентов приложения или службы зависит от взаимодействия с контейнерами и знания того, как правильно их реализовать [4]. Построение единой безопасной среды из микросервисов становится сложной задачей, а по мере увеличения их количества масштабы проблемы несопоставимо растут.

Децентрализованное хранение

Почти всегда микросервис хранит некоторое состояние. С точки зрения сервисно-ориентированных архитектур и, в частности, микросервисов, децентрализованное хранение [5] — очень важный момент. Децентрализованное хранение означает, что каждый сервис имеет свою БД и обращается исключительно к ней. Единственный случай, когда разные службы могут использовать одно хранилище,— если эти службы представляют собой точные копии друг друга. Базы данных различных сервисов друг с другом не взаимодействуют (рис. 1).

Единственный вариант взаимодействия — сетевое взаимодействие между сервисами, которое может быть реализовано как посредством API, так и через брокеры сообщений, например RabbitMQ (рис. 2).

Можно делить сервис на более мелкие до тех пор, пока ему не потребуется запросить данные из БД другого сервиса. Таким образом достигается изоляция данных — у каждого микросервиса при необходимости существует своя база данных.

Проблема множества открытых внешних адресов

Итак, допустим, у нас есть какой-то клиент который, чтобы предоставить кому-то нужные данные, взаимодействует с совокупностью других сервисов. Тогда его схема при использовании нескольких микромодулей будет следующая (рис. 3).

Казалось бы, все просто — клиент может обращаться ко всем этим сервисам. На деле получается, что конфигурация клиента слишком сильно разрастается. Но даже это не является основной проблемой в данной ситуации. Основную угрозу представляет необходимость наличия открытых внешних адресов, а также портов для каждого из микросервисов, что увеличивает площадь для кибератак, которые могут целенаправленно выводить из строя отдельные модули. В этом случае можно применить принцип API Gateway (рис. 4).

API Gateway предоставляет данные в том виде, в каком они нужны конкретно именно этому типу пользователей. Например, если будет веб и мобильное приложение, необходимо будет создать два API Gateway, которые будут собирать данные из сервисов и предоставлять их немного по-разному. API Gateway занимается лишь передачей данных и ни в коем случае не со-

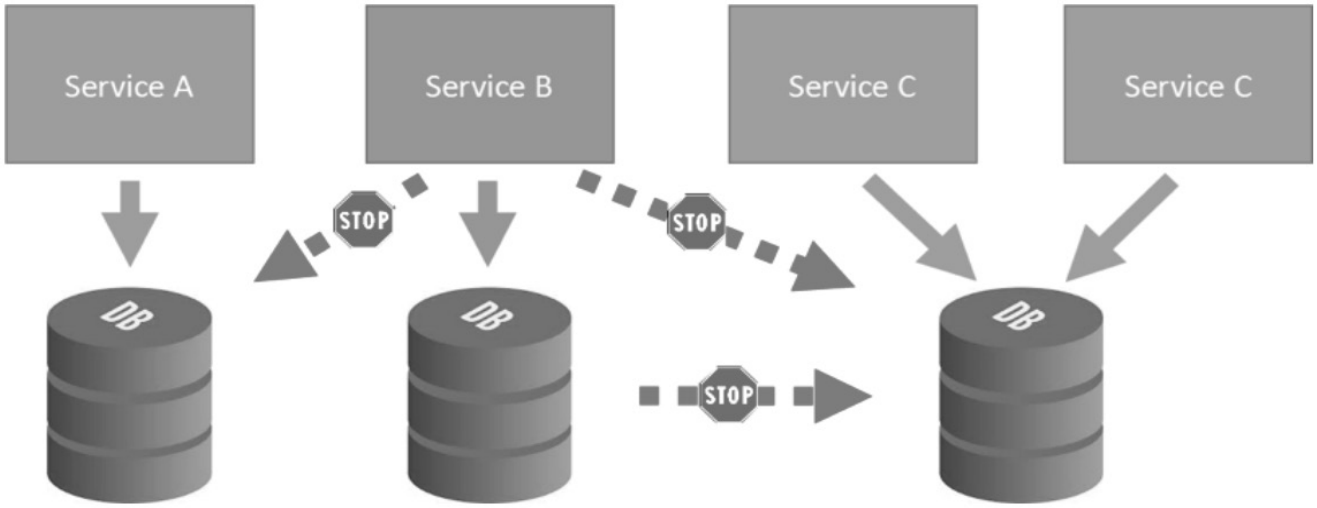

Рис. 1. Децентрализованное хранение

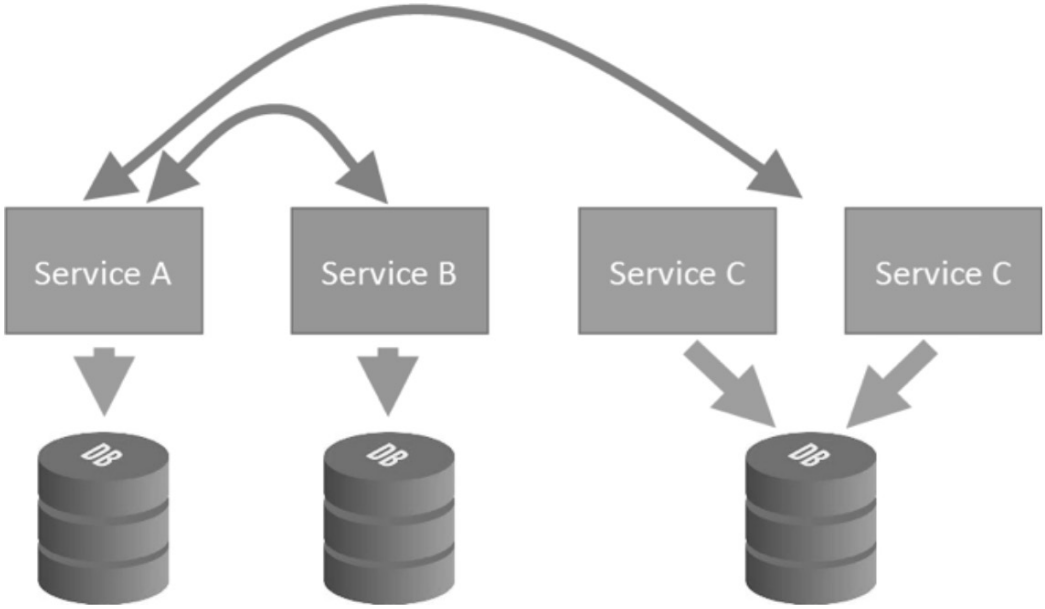

Рис. 2. Передача данных между микросервисами

держит никакой серьезной бизнес-логики, иначе бы эта логика везде дублировалась, и, как следствие, эту логику сложно поддерживать.

Express Gateway

Существует несколько возможных популярных реализаций, одна из таких — Express Gateway [6] — это API-интерфейс для микросервисов, созданный на основе фреймворка ExpressJS. Благодаря ему возможно настраивать маршрутизацию запросов. Например, чтобы все запросы на адрес https://handwriter.ru/api/store/file перенаправлялись на внутренний сервис 192.168.7.77:3002/file, где адрес https://handwriter.ru/api является единственной точкой входа в систему. Важно отметить, что такие сервисы позволяют редактировать и дополнять входные данные, например, интересным решением будет интеграция сервиса аутентификации в сам API Gateway. Тогда остальные сервисы в случае успешно пройденной аутентификации будут получать токен сессии, который был добавлен в заголовки запроса выше.

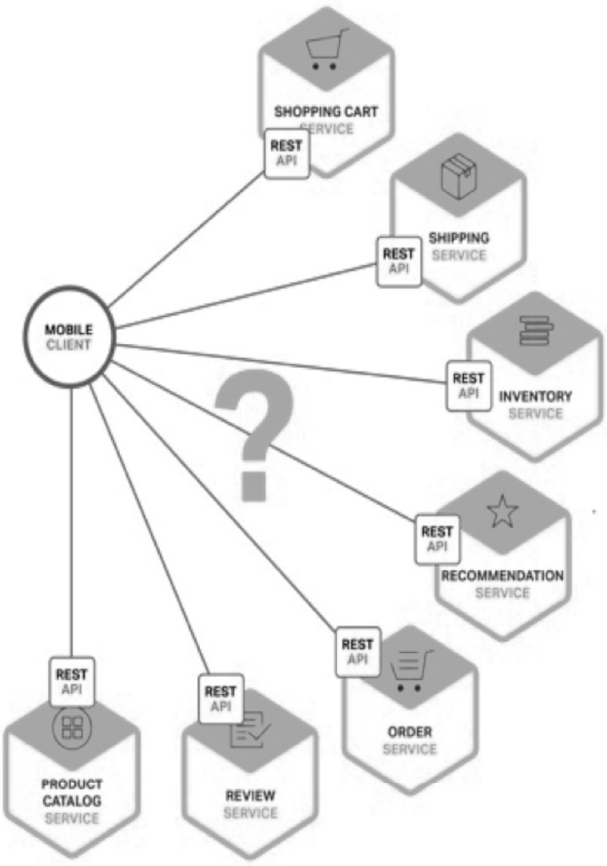

Рис. 3. Неправильный вариант распределения API

Пример конфигурации для Express Gateway

```
http:
   port: ${PORT:-8080}
https:
  port: 9443
 # hostname: localhost
apiEndpoints:
  store:
    methods: "GET, POST, PUT"
    # host: "213.183.41.33"
    paths: "/api/store*"
  converter:
    methods: "GET, POST, PUT"
    # host: localhost
    paths: "/api/converter*"
  auth:
    methods: "GET, POST, PUT"
   # host: localhost
    paths: "/api/auth*"
serviceEndpoints:
  storeService:
   url: "http://${HOST:-192.168.7.77}:${STORE_PORT:-3002}"
  converterService:
    url: "http://${HOST:-192.168.7.77}:${CONVERTER_PORT:-3012}"
  authService:
    url: "http://${HOST:-192.168.7.77}:${AUTH_PORT:-3098}"
  - policies:
  - basic-auth
  - cors
  - expression
  - key-auth
  - log
  - oauth2
  - proxy
  - rate-limit
  - rewrite
pipelines:
  store:
    apiEndpoints:
      - store
    policies:
      - rewrite:
        - action:
            search: "/api/store"
            replace: "/api"
      - proxy:
```

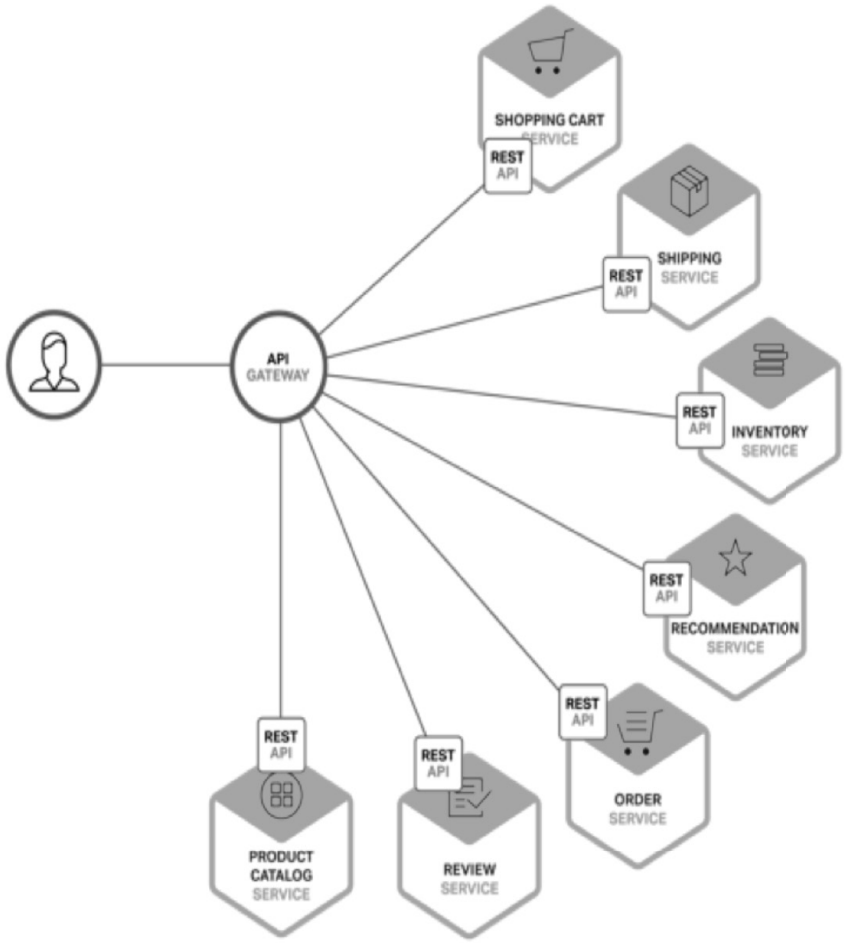

Рис. 4. Внедрение API Gateway в систему

```
        - action:
            serviceEndpoint: storeService
converter:
  apiEndpoints:
    - converter
  policies:
    - rewrite:
      - action:
          search: "/api/converter"
          replace: "/api"
    - proxy:
      - action:
          serviceEndpoint: converterService
auth:
  apiEndpoints:
    - auth
  policies:
    - rewrite:
      - action:
          search: "/api/auth"
          replace: "/api"
    - proxy:
      - action:
          serviceEndpoint: authService
```

Обратный прокси-сервер Traefik

Подобную задачу можно также решить с помощью Traefik [7], который помимо роутинга предоставляет также возможность анализа проходящего трафика, а также распределения нагрузки.

Traefik — это обратный прокси-сервер с открытым исходным кодом, обеспечивающий простую работу с микросервисами и/или контейнерами с приложениями.

Обратный прокси-сервер (reverse proxy, реверс-прокси) служит для ретрансляции запросов из внешней сети к каким-либо серверам/сервисам внутренней сети (например веб-сервера, БД или файловые хранилища) и позволяет:

- обеспечить сокрытие структуры внутренней сети и подробностей о находящейся в ней сервисах;

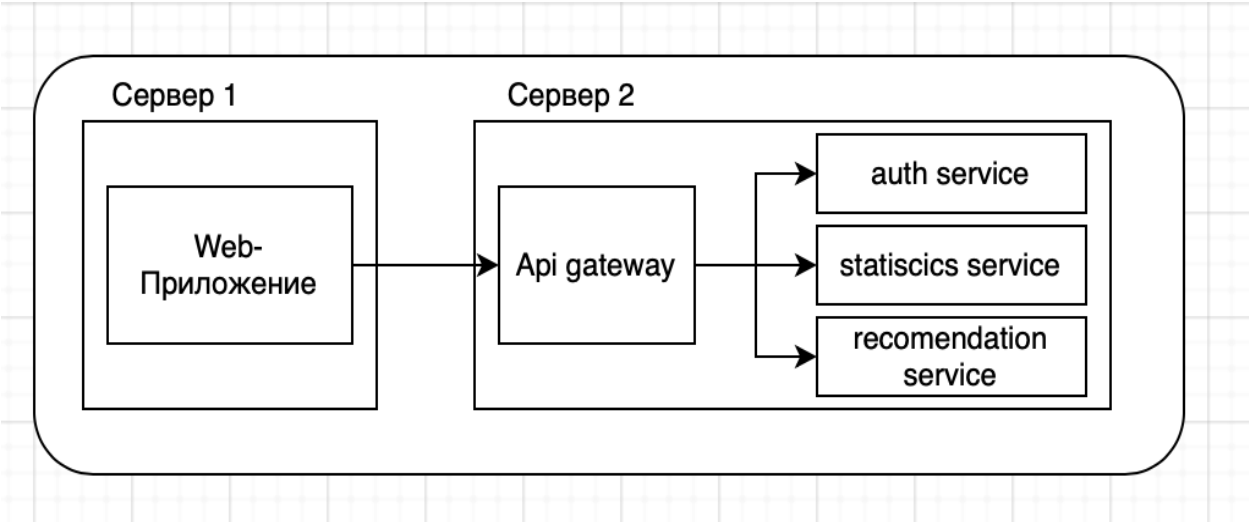

Рис. 5. Решение с API Gateway

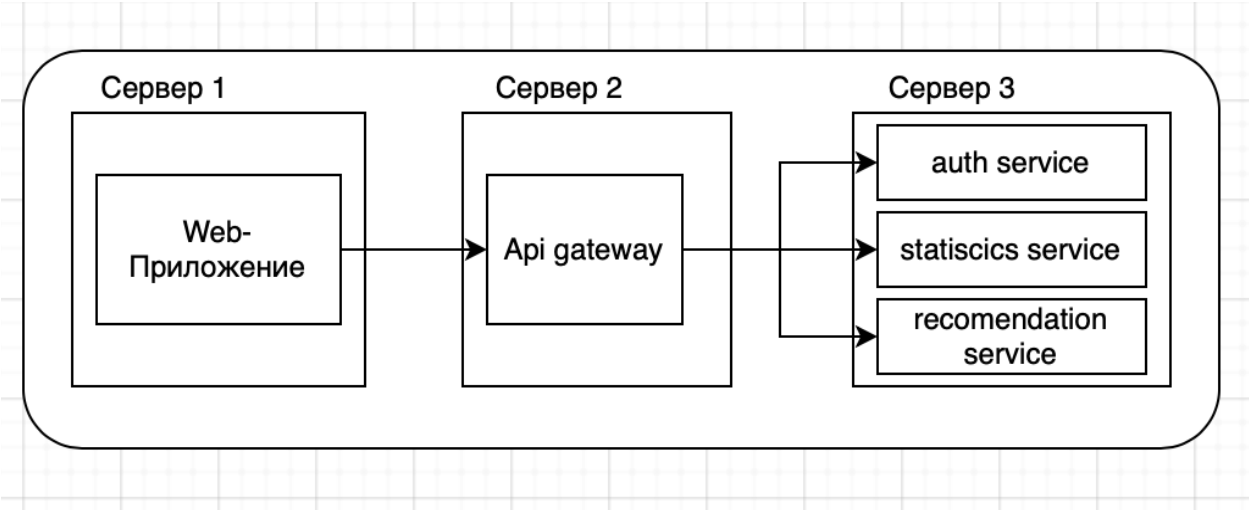

Рис. 6. Решение с API Gateway и OpenVPN

- осуществлять балансировку нагрузки (load balancing) между экземплярами одного и того же сервиса или серверами с одинаковыми задачами;
- обеспечить зашифрованное (HTTPS) соединение между клиентом и любым сервисом, в таком случае SSL сессия создается между клиентом и прокси, а между прокси и сервисом во внутренней сети устанавливается незашифрованное HTTP соединение; если сервис поддерживает HTTPS, то можно организовать зашифрованное соединение и во внутренней сети;
- организовать контроль доступа к сервисам (аутентификацию клиента), а также установить файрвол (брандмауэр).

Сокрытие IP-адреса сервисов с критичными данными

В некоторых проектах может возникнуть необходимость хранить чувствительные данные, и полностью обезопасить себя от возможного раскрытия IP-адреса.

В предыдущих случаях мы рассматривали ситуацию, когда API Gateway и микросервисы размещались на одном сервере. При такой конфигурации остается возможность подобрать данные адресов отдельных микросервисов и определить примерное нахождение сервера.

Чтобы решить эту проблему, можно перенести сервисы на отдельно выделенное оборудование, адрес

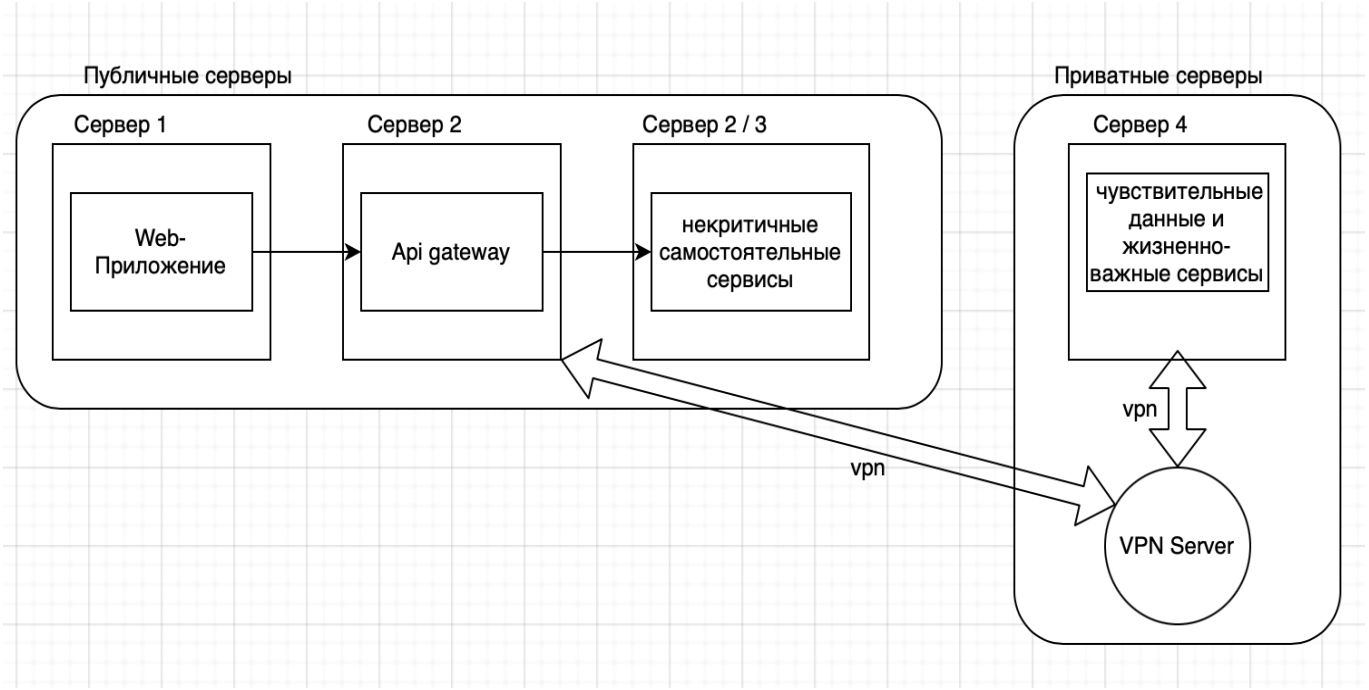

Рис. 7. Решение с API Gateway, OpenVPN и приватными серверами

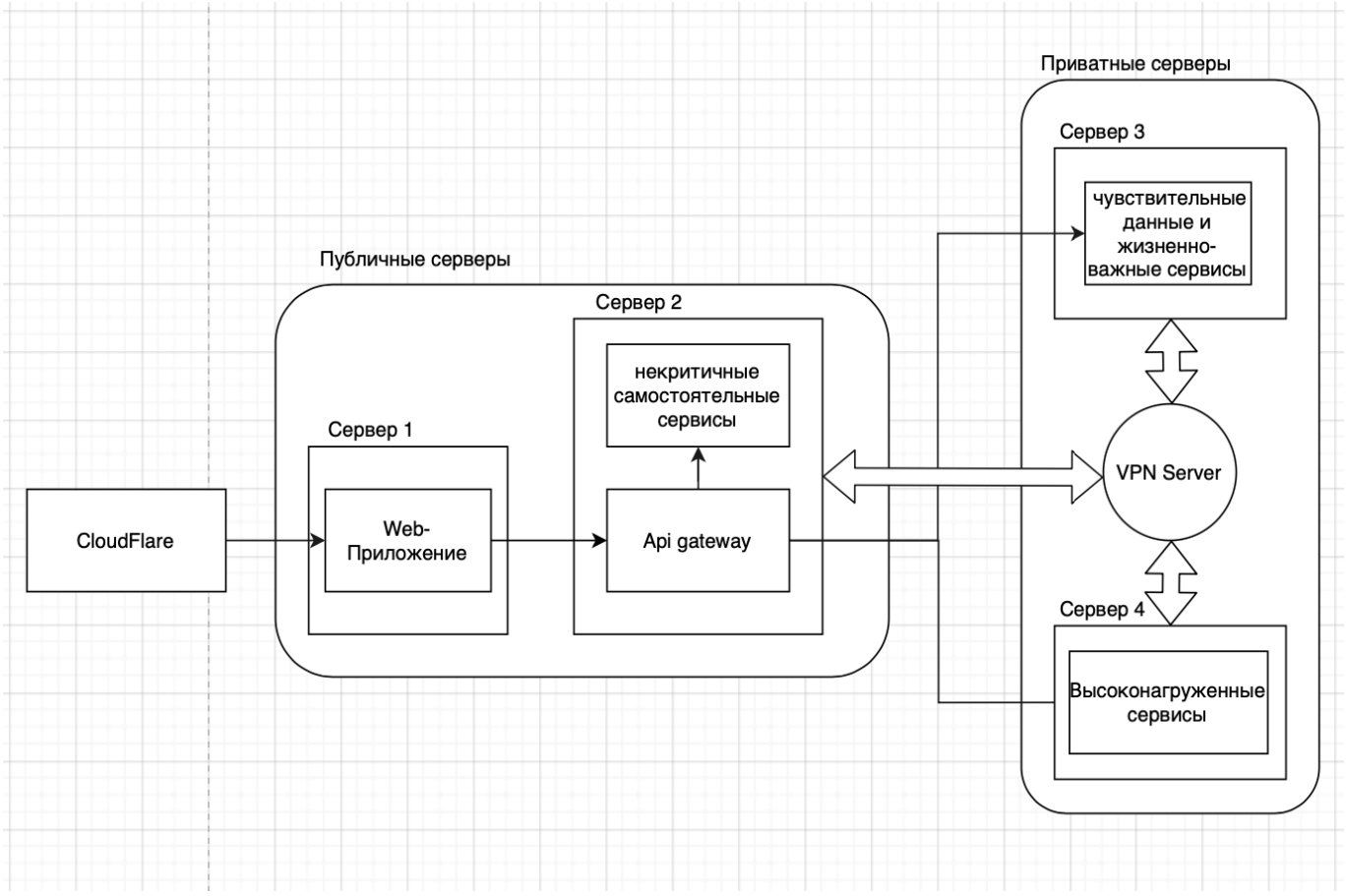

Рис. 8. Решение с API Gateway, OpenVPN, приватными серверами и Cloudflare

которого будет известен только внутри сервера, где развернут Gateway. Получим следующую схему (рис. 6).

В данной реализации проходная способность системы будет зависеть от сервера-посредника (где расположен Gateway). Однако сервер 3 все еще вынужден иметь открытые порты, так как мы будем напрямую на него посылать запросы с сервера 2. Чтобы изолировать чувствительные данные и жизненно важные сервисы от возможных атак и утечек в интернет, стоит заблокировать доступ извне для всех портов на оборудовании, где будет храниться важная информация. Для решения добавим в систему VPN сервер, который объединит наш Gateway через виртуальную приватную сеть со скрытым ранее сервером и позволит им общаться между собой по защищенному каналу связи. Одним из наиболее популярных реализаций VPN является OpenVPN [8], который активно развивается и является open source проектом.

Тогда получим следующую схему (рис. 7).

Исходя из личного опыта, можно отметить, что современное оборудование позволяет размещать VPN сервер даже на маршрутизаторах и строить сложные защищённые сети.

Защита от DDoS-атак

Цель DDoS-атаки [9] — добиться отказа в обслуживании подключенных к Интернету устройств: сетевого оборудования и инфраструктуры, различных интернет-сервисов, веб-сайтов и веб-приложений, инфраструктуры Интернета вещей.

Подавляющее большинство атак развиваются в следующей последовательности:

1. сбор данных о жертве и их анализ с целью выявления явных и потенциальных уязвимостей, выбор метода атаки;
2. подготовка к атаке путем развертывания вредоносного кода на компьютерах и подключенных к Интернету устройствах, управление которыми удалось перехватить;
3. генерация потока вредоносных запросов с множества устройств, находящихся под управлением злоумышленника;
4. анализ результативности атаки: если целей атаки добиться не удалось, злоумышленник может провести более тщательный анализ данных и выполнить повторный поиск методов атаки (переход к п. 1).

CloudFlare

На сегодняшний день на рынке представлено множество решений для защиты от DDoS атак, одно из них — CloudFlare.

CloudFlare DDoS Protection [10] — сервис, предоставляющий услуги по защите веб-сайтов от DDoS-атак. Решение выполняет фильтрацию трафика через свои центры перед тем, как он будет направлен на сайт заказчика, позволяет защитить его данные благодаря надежной инфраструктуре CloudFlare и очень компетентной команде специалистов. Многоуровневый подход CloudFlare DDoS Protection к обеспечению безопасности сочетает в себе несколько возможностей по предотвращению последствий DDoS-атак в одном сервисе. Он предотвращает сбои, вызванные вредоносным трафиком, позволяя при этом пропускать легитимный, сохраняя веб-сайты, приложения и API высокодоступными и производительными.

Результатом интеграции CloudFlare стала следующая система, которая уже успешно реализована и используется в существующем проекте (рис. 8).

Заключение

В результате разработанной схемы была реализована устойчивая и безопасная среда для микросервисного веб-приложения, в котором применены все описанные в статье решения. На отдельном сервере настроена виртуальная приватная сеть (VPN), которая является основным шлюзом системы и позволяет получать доступ к закрытым адресам. А все запросы фильтруются и перенаправляются с помощью обратного прокси-сервера Traefik, он же и занимается распределением нагрузки между модулями — микросервисами. Также подключен Cloudflare, который предоставляет услуги сети CDN (Content Delivery Network), которая ускоряет доставку контента до пользователей и дополнительно защищает от зловредного трафика и DDoS-атак.

ЛИТЕРАТУРА

1. How to Exploit a Microservice Architecture https://dzone.com/articles/how-to-exploit-a-microservice-architecture (Дата обращения: 5.11.2021)
2. Hacking Your Way Through Microservice Architecture https://hackernoon.com/hacking-your-way-through-microservice-architecture-8o1c34pa (Дата обращения: 7.11.2021)
3. Microservices Security: Challenges and Best Practices https://www.neuralegion.com/blog/microservices-security/ (Дата обращения: 4.11.2021)

4. Fundamental microservices security best practices https://searchapparchitecture.techtarget.com/tip/4-fundamental-microservices-security-best-practices (Дата обращения: 9.11.2021)
5. Микросервисная архитектура: характерные особенности, достоинства и недостатки https://www.tadviser.ru/index.php/Статья:_Микросервисная_архитектура:_характерные_особенности,_достоинства_и_недостатки (Дата обращения: 1.11.2021)
6. Express Gateway — A microservices API Gateway built on top of Express.js https://survivejs.com/blog/express-gateway-interview/ (Дата обращения: 9.11.2021)
7. Использование Traefik в качестве обратного прокси для контейнеров Docker в Ubuntu 18.04 https://www.digitalocean.com/community/tutorials/how-to-use-traefik-as-a-reverse-proxy-for-docker-containers-on-ubuntu-18–04-ru (Дата обращения: 12.11.2021)
8. How To Set Up and Configure an OpenVPN Server on CentOS https://www.digitalocean.com/community/tutorials/how-to-set-up-and-configure-an-openvpn-server-on-centos-7 (Дата обращения: 11.11.2021)
9. Что такое DDoS (Distributed Denial of Service). Как защититься от DDoS-атак. https://stormwall.pro/knowledge-base/termin/ddos-protection (Дата обращения: 3.11.2021)
10. CloudFlare DDoS Protection https://www.anti-malware.ru/products/cloudflare-ddos-protection (Дата обращения: 11.11.2021)

Московский авиационный институт