УДК 004.8
DOI: 10.26102/2310-6018/2026.59.8.009

Анализ производительности и отказоустойчивости системы при переходе на архитектуру микросервисов

Л.А. Коробова[1], И.А. Матыцина[2]✉, А.В. Калач[3], Ю.Н. Золотарев[2], Д.Н. Никитин[2]

[1]Компания "Технопарк-В", Воронеж, Российская Федерация
[2]Воронежский институт ФСИН России, Воронеж, Российская Федерация
[3]Воронежский государственный университет инженерных технологий, Воронеж, Российская Федерация

Резюме. На современном этапе развития языков программирования нестандартной задачей является задача компоновки программных файлов и классов между собой. Указанные структурные единицы можно рассматривать, как обособленные элементы сервисов. Этим занимаются программисты для разграничения мест расположения исполняемых файлов. В рамках исследования рассмотрены некоторые подходы к созданию программных систем, начиная с монолитной структуры. Приведены ее достоинства и недостатки. Поиск новых форм обоснован прогрессом логики в программных модулях разработчиков, связанных с улучшением результативности, выстраиванием связей программ и расширением масштабируемости высоконагруженных систем. Как правило, любое приложение состоит из множества небольших независимых программных продуктов. Языки программирования используют определенный архитектурный стиль, основанный на разработке автономных и слабо связанных программных модулей. Это легло в основу возникновения микросервисной архитектуры. Разработчики и программисты к вопросам архитектуры систем подходят с осторожностью. Представленные в статье результаты являются итогом исследований как построения языковых структур, так и самих приложений. Микросервисность уже давно лежит в основе языков программирования. А при построении приложений, особенно больших и монолитных, рекомендуется использовать различные узкие блоки, как функционал микросервисной архитектуры. Это позволяет приложению использовать многопоточность и тем самым увеличивать скорость выполнения задачи. Кроме этого, при увеличении нагрузки на систему срабатывает масштабируемость и отказоустойчивость. Для понимания логики работы приложения в работе исследовано приложение решения задачи расчета кратчайшего расстояния между точками тремя методами. Приложение было масштабировано на *n*-экземпляров приложения. Отказоустойчивость подтверждена тем, что при выходе одного сервиса из строя нагрузка распределится на остальные. Использован язык Java с надстройкой SpringBoot.

Ключевые слова: микросервисная архитектура, масштабируемость, отказоустойчивость, высоконагруженные системы, задача коммивояжера, Java.

Для цитирования: Коробова Л.А., Матыцина И.А., А.В. Калач, Золотарев Ю.Н., Никитин Д.Н. Анализ производительности и отказоустойчивости системы при переходе на архитектуру микросервисов. *Моделирование, оптимизация и информационные технологии.* 2026;14(8). URL: https://moitvivt.ru/ru/journal/article?id=2572 DOI: 10.26102/2310-6018/2026.59.8.009

Analyzing system performance and fault tolerance when migrating to a microservices architecture

L.A. Korobova[1], I.A. Matytsina[2]✉, A.V. Kalach[3], Yu.N. Zolotarev[2], D.N. Nikitin[2]

[1]Technopark-V Company, Voronezh, the Russian Federation
[2]Voronezh Institute of the Federal Penitentiary Service of Russia, Voronezh, the Russian Federation

[3]*Voronezh State University of Engineering Technologies, Voronezh, the Russian Federation*

Abstract. At the current stage of programming language development, composing program files and classes is a non-standard task. These structural units can be viewed as isolated service elements. Programmers use this approach to delimit the locations of executable files. This study examines several approaches to creating software systems, starting with a monolithic structure. Its advantages and disadvantages are discussed. The search for new forms is justified by advances in logic in developer software modules related to improving performance, building program connections, and expanding the scalability of high-load systems. Typically, any application consists of many small, independent software products. Programming languages employ a specific architectural style based on the development of autonomous and loosely coupled software modules. This formed the basis for the emergence of microservice architecture. Developers and programmers approach system architecture issues with caution. The results presented in this article are the result of research into both the construction of language structures and the applications themselves. Microservices have long been a foundation of programming languages. When building applications, especially large and monolithic ones, it is recommended to use various narrow blocks, such as those found in microservice architecture. This allows the application to utilize multithreading, thereby increasing task execution speed. Furthermore, scalability and fault tolerance are activated as the system load increases. To understand the application's logic, this paper examined an application solving the problem of calculating the shortest distance between points using three methods. The application was scaled to n instances. Fault tolerance was confirmed by the fact that if one service fails, the load is distributed to the others. Java with the SpringBoot framework was used.

Keywords: microservice architecture, scalability, fault tolerance, high-load systems, traveling salesman problem, Java.

For citation: Korobova L.A., Matytsina I.A., Kalach A.V., Zolotarev Yu.N., Nikitin D.N. Analyzing system performance and fault tolerance when migrating to a microservices architecture. *Modeling, Optimization and Information Technology.* 2026;14(8). (In Russ.). URL: https://moitvivt.ru/ru/journal/article?id=2572 DOI: 10.26102/2310-6018/2026.59.8.009

Введение

При написании любой программы перед разработчиком встает первоочередная задача, какую архитектуру выбрать для информационной системы. Развитие бизнес-логики, сторонних библиотек делали информационную систему большой и неповоротливой. Чтобы сделать какую-либо доработку, приходилось перебирать множество зависимых компонентов. Бизнес страдал этой проблемой в виде потери денежных средств на поддержку таких «монстров». Вместе с развитием информационной науки были выдвинуты ключевые правила разделения частей и настраивания коммуникации в виде абстракции и единых протоколов взаимодействия [1, 2]. Гораздо удобнее в системе менять части в виде модулей, а не переписывать всю систему. Такие правила стали называть микросервисной архитектурой, где сервис – структурная единица, которая делает одно «важное действие».

Монолит практически не масштабируется, в нем сложно добиться отказоустойчивости. При промышленной нагрузке на программное обеспечение монолит выйдет из строя. Микросервис поддерживает горизонтальное и вертикальное масштабирование.

В статье приведено обоснование разработки программного обеспечения в микросервисной архитектуре с аргументами. Обосновано, почему сервисы легко масштабировать, почему данная архитектура отказоустойчива и высокопроизводительна [3, 4]. Исследуется информационная система с использованием современного стека и технологий. Главная задача приложения – расчет наикратчайшего пути. В основе лежат

3 метода: полный перебор, ветвей и границ, имитации отжига. Тип архитектуры – микросервис [8].

Материалы и методы

Обоснование выбора микросервисной архитектуры. Представленная архитектура типичная для высоконагруженных систем. Из-за закрытости используемого стека мало информации о таких разработках в связи с дороговизной разработок и коммерческой тайной. При необходимости развернуть *n*-экземпляров приложения, деплой произойдет буквально в 2 нажатия, благодаря оркестрации Kubernetes. Инстансы смогут параллельно или последовательно обрабатывать все https-запросы от клиентов (что обеспечит удобство масштабирования). Если один сервис упадет, запрос подхватит в работу другой экземпляр без потери данных – это объясняет отказоустойчивость. На каждом экземпляре можно включить разный алгоритм расчета, что улучшает производительность. Также в приложении есть докерфайл для быстроты развертки.

Технологический стек. Выбран язык Java [5] и популярный фреймворк SpringBoot [6]. Благодаря инверсии управления и внедрению зависимостей разработчик делегирует фреймворку работу по жизненному циклу классов приложения [7, 8]. Boot также удобен для быстрой конфигурации приложения [5] и автоматической настройки подключения к базе данных (БД). Фреймворк оперирует библиотеками, в которых уже предварительно настроены классы для использования [5, 8].

БД PostgreSQL. Данный вид системы управления базами данных выбран из-за удобного веб-интерфейса высокой производительности с глубокой оптимизацией процессов, а также отказоустойчивости, которая обеспечивается за счет ACID-совместимости. Также в данной системе управления базами данных уделяется внимание безопасности данных и кроссплатформенности, что позволяет работать на Linux, Windows, macOS. Инструменты CI/CD – Docker + Kubernetes характеризуются удобным UI интерфейсом по управлению подами или контейнерами [9, 10].

Компоненты системы серверной части и их взаимодействие. Все запросы осуществляются на сервер через API Rest. Паттерн проектирования – MVC. Model (модель) содержит всю бизнес-логику приложения. View (вид), отвечает за отображение данных пользователю. Все, что видит пользователь, генерируется View. В Controller (контроллер) хранится код, отвечающий за обработку действий пользователя [11, 12]. Типы БД (SQL) – это большая таблица для хранения историй расчетов, координат городов, расстояний между ними. Для удобной миграции БД используется Liquibase. Сервисы инфраструктуры: мониторинг (Prometheus, Grafana), логирование (ELK). Безопасность: аутентификация (OAuth2, JWT) и авторизация (RBAC, ABAC).

Результаты

Диаграмма последовательности. На Рисунке 1 представлена последовательность действий (workflow) по обработке запросов в информационной системе [12, 13]. Используется 3 математических метода [11]. В основе приложения лежат связанные блоки алгоритмических методов [8], обозначенные на Рисунке 1 как TSP, в расшифровке – «проблема путешествия торговца» [15]. Логика обработки запроса расчета кратчайшего пути представлена в виде диаграммы последовательности (Рисунок 1).

Шаг 1 (*Client*). Отправка запроса для обращения к клиентскому приложению, т. е. к точке входа в веб-сервис. Происходит согласование о получаемых данных.

Шаг 2 (*Gateway*, *Auth*). Проверка уникального цифрового ключа и подтверждение личности пользователь – аутентификация. Если проверка прошла успешно, то переход к шагу 3, иначе происходит возврат к новому запросу.

Шаг 3 (*DistanceEv*). Обращение к алгоритму расчета расстояний между точками по координатам. Результат в километрах в виде неориентированной матрицы [14, 15]. Переход к шагу 4.

Шаг 4 (*TSP*). Определение размера матрицы расстояний. Запрос на выбор метода решения задачи коммивояжера (метод перебора (Bruteforce); метод ветвей и границ (BranchAndBound) или метод имитации отжига (SimulatedAnnealing)) [8, 11].

Шаг 5 (*TSP*). Выбор алгоритма решения задачи и сохранение данных в соответствующих таблицах: Client – данные о том, какой клиент и когда запросил расчет; Tasks – задача по получению информации о времени запроса, статусу выполнения и имени (коду) алгоритма расчета; Points – информация всех точек на расчет с ссылкой на номер задачи; ResultResponce или Result – данные расчета, включающие минимальное расстояние, время выполнения расчета, json ответа для клиента.

Шаг 6 (*DB*). Сохранение результата решения (маршрут и расстояние) в базу данных через запрос INSERT INTO results (task_data).

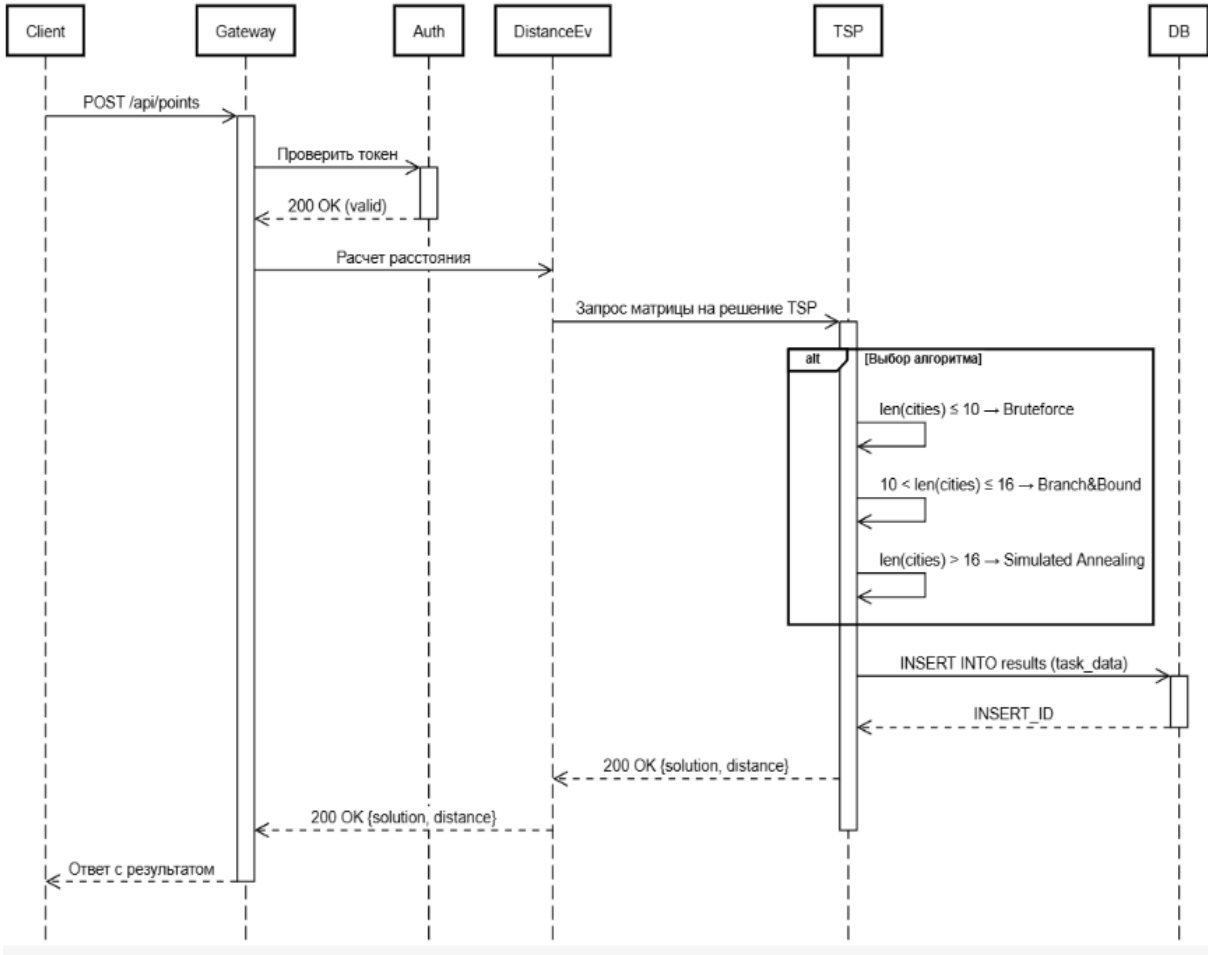

Рисунок 1 – Диаграмма последовательности
Figure 1 – Sequence diagram

Шаг 7 (*DB*, *TSP*, *DistanceEv*, *Auth*). Возврат INSERT_ID, и последовательное отправление ответа 200 OK с решением (solution) и расстоянием (distance).

Шаг 8 (*Gateway*). Направление окончательного ответа клиенту (*Client*).

Данный процесс демонстрирует типичную последовательность шагов для обработки запросов, включающий аутентификацию, выбор алгоритма на основе входных данных, сохранение результатов и возврат ответа клиенту.

Обсуждение

Диаграмма компонентов (Рисунок 2) предназначена для описания структуры системы, её модулей и их взаимосвязей [13, 14].

Анализ диаграммы

Компоненты (сервисы, микросервисы, БД или др. части системы):

– Client представляет клиентское приложение, которое отправляет запросы на сервер. В дальнейшем будет написано на ReactJs.

– Gateway: API Gateway, сервис маршрутизирует запросы к соответствующим сервисам.

– Auth: сервис аутентификации, проверяющий токены.

– DistanceEv: сервис для вычисления матрицы расстояний между точками.

– TSP: сервис решения задачи коммивояжера.

– DB: база данных, где хранятся результаты обработки и вспомогательные данные для историчности и вычислений.

Интерфейсы указывают, какие методы или функции доступны для взаимодействия с компонентами:

– validateToken(): метод в компоненте Auth для проверки токена;

– sendRequest(): метод в компоненте Client для отправки HTTP-запроса;

– calculateDistance(): метод в компоненте DistanceEv для вычисления матрицы расстояний;

– storeResult(): метод в компоненте DistanceEv для сохранения результата;

– solveTSP(): метод в компоненте TSP для решения задачи коммивояжера;

– queryMatrix(): метод в компоненте DB для получения матрицы из базы данных;

– insertResult(): метод в компоненте DB для сохранения результата в БД.

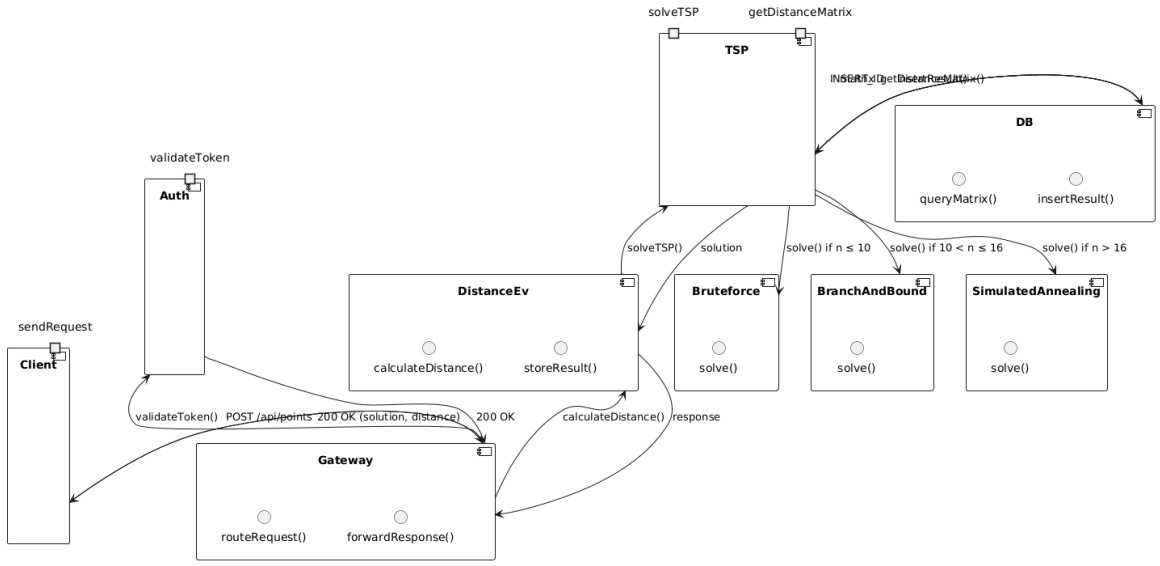

Рисунок 2 – Диаграмма компонентов
Figure 2 – Component diagram

Взаимодействие осуществляется по линиям связи:

– Client→Gateway: клиент отправляет POST-запрос (POST /api/points) через Gateway;

– Gateway→Auth: Gateway перенаправляет запрос на Auth для проверки токена;

– Auth→Gateway: после успешной аутентификации, Auth возвращает 200 OK;

– Gateway→DistanceEv: Gateway передает запрос на DistanceEv для вычисления матрицы расстояний;

– DistanceEv→TSP: DistanceEv передает матрицу расстояний в TSP для решения задачи коммивояжера;

– TSP→Bruteforce/BranchAndBound/SimulatedAnnealing: TSP выбирает подходящий алгоритм для решения задачи в зависимости от количества городов [8, 11];

– TSP→DB: результат решения сохраняется в БД через метод insertResult();

– DB→TSP: при необходимости, TSP может получить данные из БД через метод queryMatrix().

Возможные ответы (отклики) системы. После выполнения всех операций, TSP возвращает решение (solution и distance) обратно в Gateway. Gateway формирует финальный ответ (200 OK) и отправляет его клиенту.

Если необходимо добавить новый алгоритм в сервис решения задачи (TSP), например, добавить генетический алгоритм (GeneticAlgorithm), есть возможность интегрировать в диаграмму Рисунка 2 дополнительный компонент, подобно Bruteforce, BranchAndBound и SimulatedAnnealing [11, 13]. Результат развертывания дополнительных компонент в виде диаграммы показан на Рисунке 3.

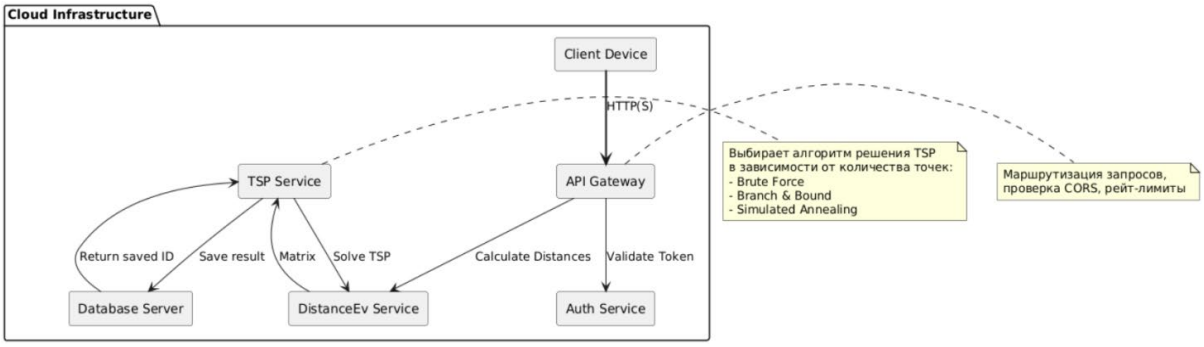

Рисунок 3 – Диаграмма развертывания
Figure 3 – Deployment diagram

Анализ диаграммы развертывания. CloudInfrastructure. Это общая структура облачной инфраструктуры, где все компоненты взаимодействуют друг с другом. Серверная и клиентская часть разворачивается в докер контейнерах. Контейнеры деплоются в поды Kubernetes.

Client Device. Это фронт приложение пользователя, которое отправляет запросы на сервер через HTTPS 443 (выходящий) или HTTP (порт 80). Запросы могут быть как GET, так и POST, например, для решения задачи TSP или получения результатов.

API Gateway. Является центральным входным узлом для всех запросов от Client Device. Внешний входной пункт, слушает 443 (HTTPS) и/или 80 (HTTP). Маршрутизация запросов: определяет, какой сервис должен обработать запрос. Проверка CORS (Cross-OriginResourceSharing): обеспечивает безопасность при работе с кросс-доменными запросами. Рейтинговые лимиты (RateLimits): контролирует количество запросов от одного клиента, чтобы предотвратить перегрузку системы.

AuthService. Сервис аутентификации и авторизации. Внутренний сервис проверки токена использует HTTP и порт 3001. Получает токен из запроса. Проверяет валидность токена. Если токен действителен, разрешает доступ к ресурсам. Если токен недействителен, возвращает ошибку.

DistanceEvService. Часть общего сервиса delivery-planner. Вычисляет расстояния между точками. Передает вызов TSP Service. Возвращает статус http 200 или 500 с пояснением ошибки клиенту. Использует http с портом 8080.

TSP Service. Часть общего сервиса delivery-planner. Принимает матрицу от DistanceEv Service. Возвращает статус http 200 или 500 с пояснением ошибки DistanceEv. Использует http с портом 8080.

DatabaseServer. Хранит данные о решении задачи TSP и различную вспомогательную информацию. При необходимости возвращает ID сохраненного результата обратно в TSP Service. Использует протокол PostgreSQL и порт 5432.

Все внутренние сервисы (Auth, DistanceEv, TSP и др.) могут быть доступны только через внутреннюю сеть облака (VPC), а не напрямую из интернета. Для контейнерных систем (Docker/Kubernetes) эти порты могут быть проброшены в контейнеры или использоваться в docker-compose.yml или манифестах Kubernetes [6].

Заключение

Микросервисная архитектура, представленная в данной статье, демонстрирует преимущества перед монолитом, особенно в контексте высоконагруженных, масштабируемых и отказоустойчивых систем. Если говорить о масштабируемости, то каждый компонент системы – Auth, DistanceEv, TSP или DB – может масштабироваться независимо от других звеньев. Результат исследования: при добавлении 3-х экземпляров сервиса TSP пропускная способность системы выросла на 180 %, что подтверждает линейную масштабируемость.

Затрагивая отказоустойчивость и изоляцию сбоев, можно отметить, что в микросервисной архитектуре сбой одного компонента не приводит к падению всей системы. Например, если DistanceEv временно недоступен, то TSP может использовать кэшированную матрицу из DB. Если Auth упал, Gateway может временно разрешать доступ по IP-адресам белого списка или использовать fallback-токены. CircuitBreaker между сервисами предотвращает каскадные отказы. В монолите же при допуске ошибки в одном модуле обрушится весь кластер приложений.

Как показано на Рисунке 1, для решения задачи TSP используются разные алгоритмы (Bruteforce, BranchAndBound, SimulatedAnnealing), и система динамически выбирает подходящий в зависимости от размера входных данных [11]. Это доказывает гибкость архитектуры. При добавлении нового алгоритма (например, GeneticAlgorithm) требуется только создание нового компонента или модуля внутри TSP и регистрация его в маршрутизаторе алгоритмов. Все это возможно без пересборки всей системы и без остановки других сервисов. В монолите это невозможно без полного редеплоя и риска сломать другие части ИС.

Технологическая независимость предполагает, что каждый микросервис может быть написан на разных языках, использовать разные фреймворки, базы данных и очереди сообщений, в зависимости от задачи и применяемого алгоритма. Представленная микросервисная архитектура – это архитектурное решение, отвечающее требованиям современных высоконагруженных, распределённых систем.

СПИСОК ИСТОЧНИКОВ / REFERENCES

1. Яковлев Н.С., Михайлов А.А., Килиба Ю.В. и др. Использование современных микроконтроллеров при проектировании электронных устройств. *Вестник Новгородского государственного университета*. 2024;(3):403–415. https://doi.org/10.34680/2076-8052.2024.3(137).403-415

Yakovlev N.S., Mikhailov A.A., Kiliba Yu.V., et al. Using modern microcontrollers when designing electronic devices. *Vestnik of Novgorod State University.* 2024;(3):403–415. (In Russ.). https://doi.org/10.34680/2076-8052.2024.3(137).403-415

2. Герасимов А.Ю., Канахин А.А., Привалов П.А. и др. Применение статического анализа исходного кода для поиска проблем с производительностью: примеры из практики. *Труды Института системного программирования РАН*. 2022;34(4):7–20. (На англ.). https://doi.org/10.15514/ISPRAS-2022-34(4)-1
Gerasimov A.Yu., Kanakhin A.A., Privalov P.A., et al. Case study: Source code static analysis for performance issues detection. *Proceedings of the Institute for System Programming of the RAS.* 2022;34(4):7–20. https://doi.org/10.15514/ISPRAS-2022-34(4)-1
3. Долматов Р.А., Сараджишвили С.Э. Оптимизация масштабируемости и отказоустойчивости микросервисов с применением Kubernetes. В сборнике: *Системный анализ в проектировании и управлении: Сборник научных трудов XXVIII Международной научно-практической конференции: Часть 2, 27–29 июня 2024 года, Санкт-Петербург, Россия.* Санкт-Петербург: ПОЛИТЕХ-ПРЕСС; 2024. С. 542–547. https://doi.org/10.18720/SPBPU/2/id24-557
Dolmatov R.A., Saradgishvili S.E. Optimization of Scalability and Fault Tolerance of Microservices With Using Kubernetes. In: *Systems Analysis in Design and Management: Proceedings of the XXVIII International Scientific and Practical Conference: Part 2, 27–29 June 2024, Saint Petersburg, Russia*. Saint Petersburg: POLITEKH-PRESS; 2024. P. 542–547. (In Russ.). https://doi.org/10.18720/SPBPU/2/id24-557
4. Кузнецов И.А. Оптимизация распределенных систем для мобильных приложений: улучшение производительности и масштабируемости. *Инновационная наука*. 2024;(5-1):52–57.
Kuznetsov I.A. Optimization of Distributed Systems for Mobile Applications: Improving Performance and Scalability. *Innovation Science.* 2024;(5-1):52–57. (In Russ.).
5. Korobova L.A., Matytsina I.A., Tolstova I.S., et al. Prototype mobile application definitions fresh products based on neural network. In: *Journal of Physics: Conference Series: Volume 1902: International Conference "Applied Mathematics, Computational Science and Mechanics: Current Problems" (AMCSM 2020), 07–09 December 2020, Voronezh, Russia.* IOP Publishing; 2021. https://doi.org/10.1088/1742-6596/1902/1/012118
6. Лисовский А.В. Применение контейнерных технологий в разработке систем KYC-верификации пользователей. *Вестник науки*. 2025;3(5):1406–1410.
Lisovsky A.V. The use of container systems technologies in development of systems KYC verification users. *Science Bulletin.* 2025;3(5):1406–1410. (In Russ.).
7. Бондаренко А.С., Зайцев К.С. Управление контейнерами при построении распределенных систем с микросервисной архитектурой. *International Journal of Open Information Technologies*. 2023;11(8):17–23.
Bondarenko A.S., Zaytsev K.S. Using container management systems to build distributed cloud information systems with microservice architecture. *International Journal of Open Information Technologies.* 2023;11(8):17–23. (In Russ.).
8. Артюхин В.В., Батурин Д.В., Егунова А.И. и др. Сравнительный анализ архитектурных паттернов микрофронтендов для высоконагруженных веб-платформ. *Инженерный вестник Дона*. 2026;(1). URL: https://www.ivdon.ru/ru/magazine/archive/n1y2026/10684

Artuhin V.V., Baturin D.V., Egunova A.I., et al. Comparative analysis of micro-frontend architectural patterns for high-load web platforms. *Engineering Journal of Don.* 2026;(1). (In Russ.). URL: https://www.ivdon.ru/en/magazine/archive/n1y2026/10684

9. Зиборев А.В. Антипаттерны построения микросервисных приложений в высоконагруженных проектах. *Universum: технические науки*. 2023;(11-1):29–34. https://doi.org/10.32743/UniTech.2023.116.11.16204
Ziborev A.V. Antipatterns of building microservice applications in high-load projects. *Universum: Technical Sciences.* 2023;(11-1):29–34. (In Russ.). https://doi.org/10.32743/UniTech.2023.116.11.16204
10. Коновалов Г.Г. Анализ применения микросервисной архитектуры при разработке веб-приложений. *Тенденции развития науки и образования*. 2023;(104-14):38–41. https://doi.org/10.18411/trnio-12-2023-771
11. Москалев В.А., Коробова Л.А. Реализация и анализ скорости работы алгоритмов решения задачи коммивояжёра. *Инженерные технологии.* 2025;(3):24–34.
Moskalev V.A., Korobova L.A. Implementation and analysis of the performance speed of algorithms solving the traveling salesman problem. *Engineering Technologies.* 2025;(3):24–34. (In Russ.).
12. Авцинов И.А., Емельянов А.Е., Ивлиев М.Н. Исследование влияния буферизации данных на качество управления в сетевых системах. *Вестник ТГТУ*. 2019;25(1):63–71. https://doi.org/10.17277/vestnik.2019.01.pp.063-071
Avtsinov I.A., Emelyanov A.E., Ivliev M.N. Investigating the Effect of Data Buffering on the Quality of Network Control Systems. *Transactions TSTU.* 2019;25(1):63–71. (In Russ.). https://doi.org/10.17277/vestnik.2019.01.pp.063-071
13. Авсеева О.В., Сафонова Ю.А., Черняева С.Н. Решение задачи управления качеством многошаговых вероятностных процессов с последовательно взаимосвязанными операциями. *Экономика и менеджмент систем управления*. 2016;(3-2):259–265.
Avseeva O.V., Safonova Y.A., Chernyaeva S.N. Solving Mathematical Problem of Quality Management Multistage Probabilistic Process With a Sequence of Related Operations. *Economics and Management of Control Systems.* 2016;(3-2):259–265. (In Russ.).
14. Альфара А.Ю.А., Королев Д.В., Зайцев К.С. и др. Разработка системы мониторинга для серверного приложения. *International Journal of Open Information Technologies*. 2023;11(8):24–31.
Alfara A.Yu.A., Korolev D.V., Zaytsev K.S., et al. Development of a monitoring system for a server application. *International Journal of Open Information Technologies.* 2023;11(8):24–31. (In Russ.).
15. Матыцина И.А., Баркалов В.В., Науменко В.С. Мультисверточная нейросеть для определения сигналов акселерометра и анализа производственной активности. *Известия Тульского государственного университета. Технические науки*. 2024;(10):299–302.
Matytsina I.A., Barkalov V.V., Naumenko V.S. Multi-convolutional neural network for determining accelerometer signals and analyzing production activity. *News of the Tula State University. Technical Sciences.* 2024;(10):299–302. (In Russ.).

ИНФОРМАЦИЯОБАВТОРАХ / INFORMATION ABOUT THE AUTHORS

Коробова Людмила Анатольевна, кандидат технических наук, доцент, инженер-программист, Компания «Технопарк-В», Воронеж, Российская Федерация.
e-mail: lyudmila_korobova@mail.ru
ORCID: 0000-0003-1349-732X

Lyudmila A. Korobova, Candidate of Engineering Sciences, Docent, Software Engineer, Technopark-V Company, Voronezh, the Russian Federation.

Матыцина Ирина Александровна, кандидат технических наук, профессор кафедры безопасности информации и защиты сведений, составляющих государственную тайну, Воронежский институт ФСИН России, Воронеж, Российская Федерация.
e-mail: irina210390@mail.ru
ORCID: 0000-0001-9236-9654

Irina A. Matytsina, Candidate of Engineering Sciences, Professor at the Department of Information Security and Protection of Information Constituting a State Secret, Voronezh Institute of the Federal Penitentiary Service of Russia, Voronezh, the Russian Federation.

Калач Андрей Владимирович, доктор химических наук, профессор, заведующий кафедрой информационных технологий, моделирования и управления, Воронежский государственный университет инженерных технологий, Воронеж, Российская Федерация.
e-mail: a_kalach@mail.ru
ORCID: 0000-0002-8926-3151

Andrey V. Kalach, Doctor of Chemical Sciences, Professor, Head of the Department of Information Technology, Modeling and Management, Voronezh State University of Engineering Technologies, Voronezh, the Russian Federation.

Золотарев Юрий Николаевич, доктор технических наук, доцент, профессор кафедры безопасности информации и защиты сведений, составляющих государственную тайну Воронежский институт ФСИН России, Воронеж, Российская Федерация.
e-mail: vifsin@36.fsin.gov.ru
ORCID: 0000-0003-0582-3009

Yuri N. Zolotaryov, Doctor of Engineering Sciences, Docent, Professor at the Department of Information Security and Protection of Information Constituting a State Secret, Voronezh Institute of the Federal Penitentiary Service of Russia, Voronezh, the Russian Federation.

Никитин Дмитрий Николаевич, начальник кабинета, Воронежский институт ФСИН России, Воронеж, Российская Федерация.
e-mail: nikitin2004@mail.ru
ORCID: 0009-0003-8868-5381

Dmitry N. Nikitin, Head of the Office, Voronezh Institute of the Federal Penitentiary Service of Russia, Voronezh, the Russian Federation.

Статья поступила в редакцию 10.07.2026; одобрена после рецензирования 19.08.2026; принята к публикации 26.08.2026.

The article was submitted 10.07.2026; approved after reviewing 19.08.2026; accepted for publication 26.08.2026.