

В. К. Базанов¹✉, А. А. Шаронов¹

Проектирование и исследование архитектуры высоконагруженного веб-приложения для организации научных конференций на основе микросервисного подхода

¹Сибирский государственный университет геосистем и технологий,
г. Новосибирск, Российская Федерация
e-mail: vadim.bazanov.zip@gmail.com

Аннотация. В статье рассматривается проблематика обеспечения стабильности образовательных веб-платформ в моменты экстремальной пользовательской активности. Проанализированы критические уязвимости традиционных монолитных систем при массовой загрузке бинарных данных в период закрытия регистрации на научные конференции. Цель исследования заключается в проектировании отказоустойчивой распределенной инфраструктуры, способной нивелировать резкие скачки трафика. Реализация базируется на паттерне API Gateway, применении неблокирующего ввода-вывода средствами асинхронного фреймворка FastAPI и выносе ресурсоемких процессов в шину сообщений RabbitMQ. Результаты проведенного стресс-тестирования демонстрируют выдающуюся производительность созданного шлюза: он способен обрабатывать более 230 запросов каждую секунду без сбоев. Среднее время ответа составляет 20 миллисекунд, а количество прерванных соединений равно нулю. Созданный инфраструктурный комплекс полностью готов к контейнерному развертыванию на серверных мощностях образовательных учреждений.

Ключевые слова: микросервисная архитектура, высокие нагрузки, брокер сообщений, контейнеризация, асинхронная обработка

V. K. Bazanov¹✉, A. A. Sharapov¹

Design and research of the architecture of a high-load web application for organizing scientific conferences based on a microservice approach

¹Siberian State University of Geosystems and Technologies, Novosibirsk, Russian Federation
e-mail: vadim.bazanov.zip@gmail.com

Abstract. The article addresses the challenge of maintaining the reliability of educational web platforms when they experience high levels of user engagement. It examines the inherent weaknesses of conventional monolithic architectures during the mass upload of binary data that typically occurs when conference registration deadlines are approaching. The purpose of the study is to design a fault-tolerant distributed infrastructure capable of leveling sharp traffic spikes. The implementation is based on the API Gateway pattern, uses non-blocking I/O with the FastAPI asynchronous framework, and offloads resource-intensive processes onto the RabbitMQ messaging bus. The stress testing data confirms the outstanding throughput of the developed gateway: it can process over 230 requests every second without failures. The median response time is fixed at 20 milliseconds with a zero rate of dropped connections. The created infrastructure complex is fully ready for container deployment on the server capacities of educational institutions.

Keywords: microservice architecture, high loads, message broker, containerization, asynchronous processing

Введение

Процесс цифровизации академических форумов выявляет ряд узких мест в сетевой инфраструктуре учебных заведений [1–3]. Распределение входящего трафика здесь носит ярко выраженный пиковый характер. Студенты и исследователи склонны откладывать отправку материалов до наступления крайних сроков. В период, непосредственно предшествующий закрытию регистрации, вычислительные узлы испытывают максимальную загрузку из-за необходимости параллельного приема множества тяжеловесных электронных документов. К ним относятся отсканированные экспертные заключения и многостраничные презентации в формате PDF.

Монолитные системы при подобном сценарии эксплуатации быстро исчерпывают лимиты открытых TCP-соединений. Рабочие потоки веб-сервера простаивают, ожидая завершения дисковых операций ввода-вывода. В результате интерфейс перестает отвечать даже тем пользователям, которым требуется лишь чтение текстовой информации. Чтобы ликвидировать угрозу каскадного отказа, требуется физическое разделение пулов обработки. Главная цель текущего исследования сводится к формированию отказоустойчивой программной базы для конференц-порталов путем миграции на распределенную микросервисную топологию [4, 5].

Методы и технологии

Организация прямого доступа клиентского приложения к разрозненным серверным узлам неизбежно сопровождается критическими инфраструктурными уязвимостями. Если оставить базовые настройки проброса портов для каждого отдельного микросервиса, внешние пользователи получают техническую возможность прямого сканирования внутренних баз данных и очередей сообщений. Кроме того, жесткая привязка фронтенда к множеству различных сетевых адресов существенно усложняет алгоритмы балансировки и поддержку протоколов междоменных запросов (CORS) на стороне браузера.

Чтобы исключить подобные проблемы, программные компоненты платформы потребовалось объединить в закрытую виртуальную подсеть. Физический доступ из глобальной сети ограничивается единственной точкой входа, которая инкапсулирует в себе всю сложность внутренней топологии. Клиентский интерфейс взаимодействует с бэкендом так, будто он является классической единой монолитной системой, отправляя все HTTP-пакеты на один выделенный домен. Именно поэтому построение надежных межсервисных коммуникаций базировалось на интеграции паттерна API Gateway. Концептуальная схема взаимодействия микросервисов представлена на рис. 1.

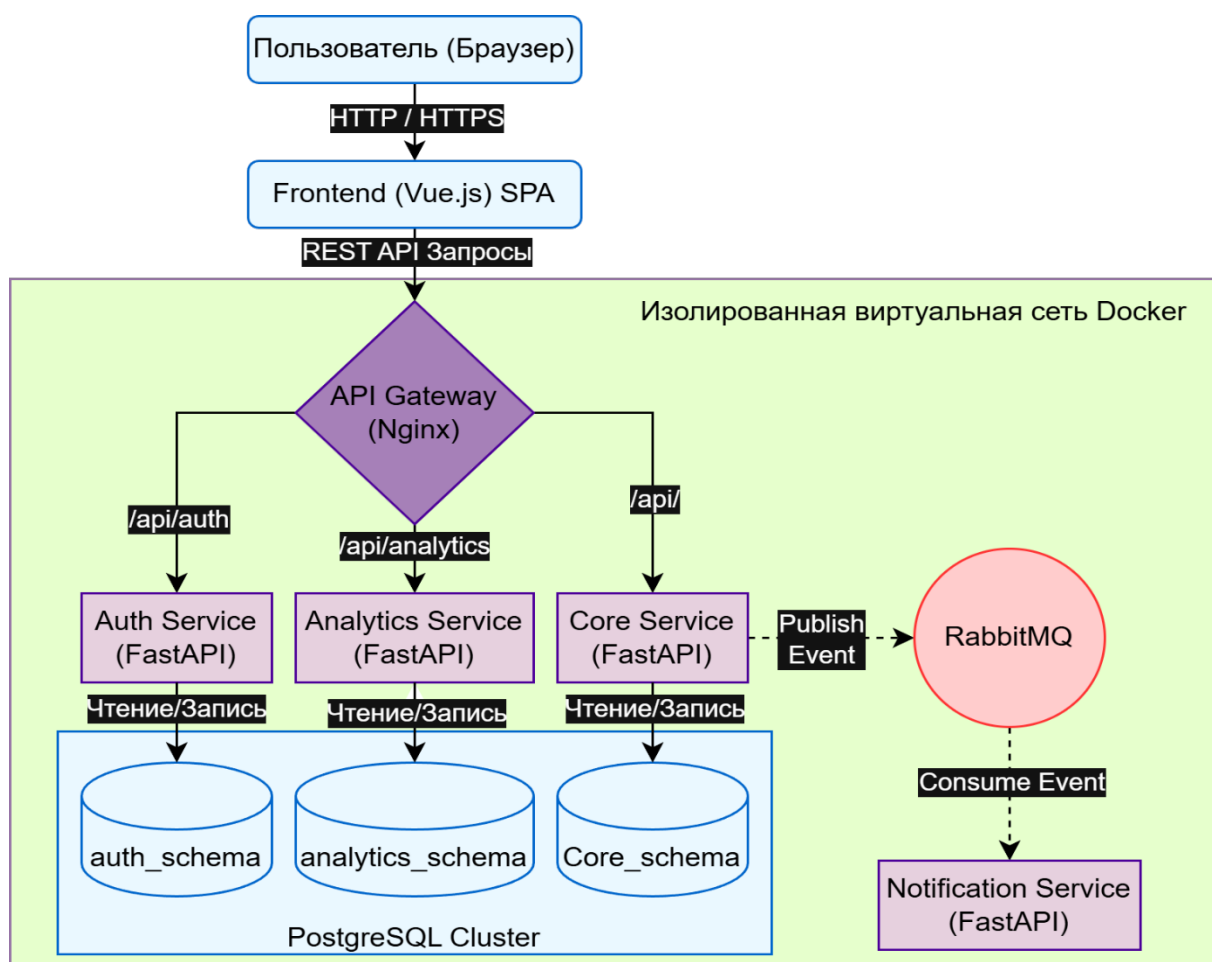


Рис. 1. Концептуальная архитектура микросервисного веб-приложения

Функции входного узла делегированы реверс-прокси серверу Nginx [6]. Программный маршрутизатор анализирует входящие заголовки и префиксы URI, после чего прозрачно перенаправляет пакеты в соответствующие изолированные контейнеры. Для предотвращения переполнения оперативной памяти бэкенда на уровне веб-сервера было внедрено строгое программное ограничение максимального размера загружаемых файлов.

Разделение серверной части произведено по принципу жесткой изоляции зон ответственности. Модуль безопасности занят исключительно криптографической проверкой хешей и выдачей Stateless-токенов стандарта JWT [7, 8]. Основной узел конференций оперирует бизнес-правилами и фиксирует метаданные заявок. Базовым инструментом программирования выбран язык Python в связке с ASGI-фреймворком FastAPI [9, 10]. Применение неблокирующего I/O позволяет серверу моментально переключаться на следующего клиента, пока ядро операционной системы выполняет сохранение байтов на накопитель.

Сглаживание пиковых скачков реализовано через шину данных. Брокер RabbitMQ выступает промежуточным буфером между приемом HTTP-запроса и реальной обработкой [11, 12]. Узел бизнес-логики формирует легковесный JSON-пакет о новом докладе, публикует его в очередь и моментально отчитывается браузеру об

успешной операции. Пример программной реализации алгоритма делегирования задачи в брокер:

```
def publish_application_event(app_id: int, user_email: str):
    message = {
        "event_type": "application_submitted",
        "payload": {
            "application_id": app_id,
            "user_email": user_email
        }
    }
    publisher.publish_message(routing_key="notifications", message=message)
```

Тяжелая логика сохранения архивов и генерации почтовых уведомлений выполняется фоновыми воркерами по мере наличия свободных тактов процессора. Развертывание всего комплекса производилось в виртуальной среде Docker [13, 14], а синтетическая нагрузка генерировалась посредством утилиты Locust [15, 16].

Результаты

Испытания пропускной способности проводились путем эмуляции активности 500 одновременных подключений. Тестирование длилось непрерывно в течение 5 минут с интенсивностью добавления (spawn rate) 10 пользователей в секунду. Скрипты тестовых агентов воспроизводили конкурентные запросы на чтение данных о секциях и параллельные попытки аутентификации с проверкой криптографических подписей. Ключевой метрикой служила фиксация деградации производительности при стрессовых воздействиях. Измеренные показатели пропускной способности шлюза приведены в табл. 1.

Таблица 1. Статистические показатели пропускной способности инфраструктуры

Тип запроса	Итерации	Отказы (Fails)	RPS	Медианное время (ms)	95-й перцентиль (ms)
POST (Авторизация)	11624	0	56,8	50	110
GET (Чтение данных)	34895	0	175,9	16	200
Агрегированный итог	46519	0	232,7	20	170

Значение медианного отклика зафиксировалось на отметке 20 миллисекунд. Локальные всплески на графике перцентилей прямо связаны с запуском процессорозависимых алгоритмов вычисления хешей при входе в систему. Примечательным

фактом стало сохранение нулевого уровня потерянных пакетов в ходе всего эксперимента (рис. 2).

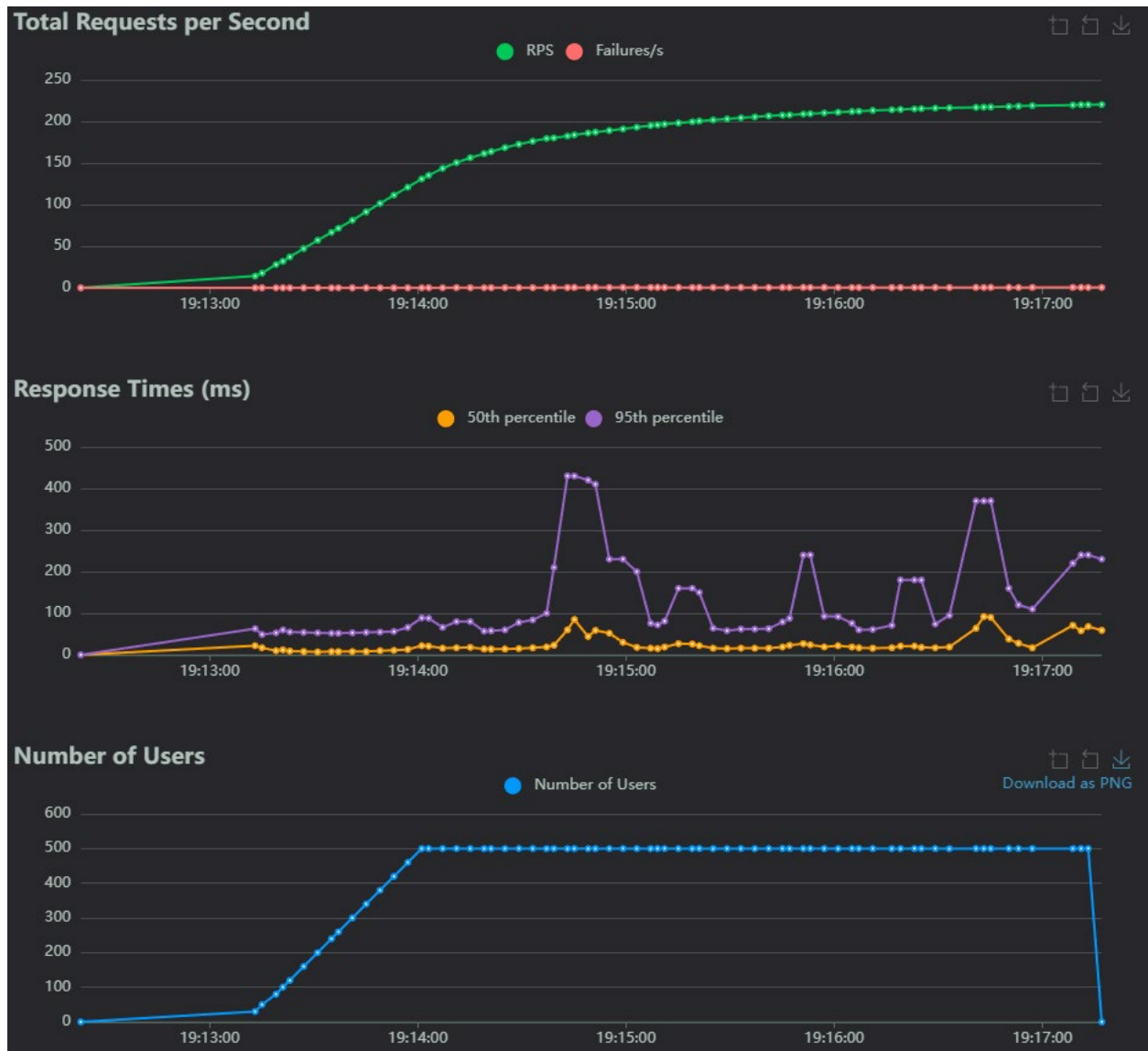


Рис. 2. Динамика времени отклика и пропускной способности при пиковой нагрузке

При обработке более сорока шести тысяч HTTP-запросов ядро системы не допустило ни одного фатального обрыва соединения или ошибки таймаута базы данных.

Обсуждение

Проанализированные метрики наглядно доказывают инфраструктурное превосходство распределенных моделей над традиционными платформами при проведении академических мероприятий [17, 18]. Монолитные продукты, используемые сейчас большинством учебных заведений, как правило, обладают ограниченным запасом эластичности при резких скачках трафика. При аналогичном объеме

конкурентных загрузок синхронные аналоги неизбежно исчерпывают пул подключений к СУБД, что влечет массовую ошибку Gateway Timeout.

Перенос логики на асинхронные рельсы позволил убрать прямую зависимость между приемом тяжелого архива и выдачей HTML-страниц другим участникам. Настроенная связка Nginx, FastAPI и RabbitMQ успешно абсорбировала тестовый удар. Зафиксированная скорость в 230 RPS многократно перекрывает реальные потребности профильных вузовских порталов в часы максимальной активности. Любые возникающие программные ошибки при такой топологии остаются внутри конкретного Docker-контейнера, не затрагивая работоспособность соседних узлов [19, 20].

Заключение

В результате исследования разработана и программно воплощена отказоустойчивая архитектура для автоматизации научных конференций. Использование принципов контейнеризации, обратного проксирования и отложенной обработки сообщений позволило сформировать сверхнадежный серверный фундамент. Использование классических монолитных подходов на финальных этапах сбора материалов зачастую сопряжено с существенной деградацией производительности веб-сервера при массовом поступлении файлов. Разработанная архитектура решает эту проблему: вынос тяжеловесных вычислений в фоновые процессы гарантировал бесперебойное обслуживание клиентских сессий. Микросервисный комплекс продемонстрировал заданные характеристики под стрессовой нагрузкой и полностью готов к развертыванию на реальных аппаратных мощностях образовательных организаций.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ньюмен С. Создание микросервисов. Проектирование и разработка / С. Ньюмен. – Санкт-Петербург : Питер, 2016. – 304 с.
2. Ричардсон К. Микросервисы. Паттерны разработки и рефакторинга / К. Ричардсон. – Санкт-Петербург : Питер, 2019. – 544 с.
3. Персиваль К. Архитектура программного обеспечения на практике / К. Персиваль. – Москва : Вильямс, 2022. – 400 с.
4. Документация фреймворка FastAPI // FastAPI. – URL: <https://fastapi.tiangolo.com> (дата обращения: 04.05.2026).
5. Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка / М. Клеппман. – Санкт-Петербург : Питер, 2018. – 640 с.
6. Фаулер М. Шаблоны корпоративных программных приложений / М. Фаулер. – Москва : Вильямс, 2018. – 544 с.
7. Стопфорд Б. Проектирование событийно-ориентированных архитектур / Б. Стопфорд. – Москва : ДМК Пресс, 2020. – 352 с.
8. Бьюнс Р. Одностраничные веб-приложения / Р. Бьюнс, С. Голдштейн. – Санкт-Петербург : Питер, 2018. – 512 с.
9. Хоффман А. Безопасность микросервисов / А. Хоффман. – Санкт-Петербург : Питер, 2021. – 352 с.
10. Фаулер М. Асинхронное программирование на Python / М. Фаулер. – Санкт-Петербург : Питер, 2023. – 416 с.
11. Моуэт Э. Использование Docker / Э. Моуэт. – Москва : ДМК Пресс, 2017. – 354 с.

12. Бейер Б. Надежность Google. Как создают архитектуру / Б. Бейер, К. Джонс, Д. Петофф. – Санкт-Петербург : Питер, 2019. – 592 с.
13. Куликов С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – Минск : Четыре четверти, 2020. – 312 с.
14. Неделку С. Nginx HTTP Server / С. Неделку. – Бирмингем : Packt Publishing, 2015. – 348 с.
15. Моргунов П. PostgreSQL. Основы языка SQL / П. Моргунов. – Санкт-Петербург : БХВ-Петербург, 2018. – 336 с.
16. Официальная документация платформы Docker // Docker Docs. – URL: <https://docs.docker.com> (дата обращения: 04.05.2026).
17. Официальная документация брокера RabbitMQ // RabbitMQ. – URL: <https://www.rabbitmq.com/documentation.html> (дата обращения: 04.05.2026).
18. Инструмент нагрузочного тестирования Locust // Locust.io. – URL: <https://locust.io> (дата обращения: 04.05.2026).
19. Официальная спецификация JSON Web Token (JWT) RFC 7519 // IETF. – URL: <https://datatracker.ietf.org/doc/html/rfc7519> (дата обращения: 04.05.2026).
20. Руководство пользователя фреймворка Vue.js // Vue.js. – URL: <https://vuejs.org/guide/introduction.html> (дата обращения: 04.05.2026).

© В. К. Базанов, А. А. Шаранов, 2026