

А. Е. Коровянский¹✉, А. А. Шарапов¹

Разработка и исследование серверной части веб-платформы управления научными конференциями с использованием REST API

¹Сибирский государственный университет геосистем и технологий,
г. Новосибирск, Российская Федерация
e-mail: lexparker925@gmail.com

Аннотация. В статье рассматривается проблема недостаточной производительности и отказоустойчивости традиционных монолитных веб-систем для управления научными мероприятиями в условиях пиковых нагрузок. Предложен и исследован подход к разработке серверной части на основе микросервисной архитектуры. Описан процесс декомпозиции системы на независимые сервисы, обоснован выбор технологического стека, включающего фреймворк FastAPI, брокер сообщений RabbitMQ и технологию контейнеризации Docker. Ключевой частью исследования является проведение и анализ результатов нагрузочного тестирования с использованием инструмента Locust. Экспериментально доказано, что предложенная архитектура после оптимизации способна стабильно обрабатывать нагрузку до 250 запросов в секунду при времени отклика, не превышающем 450 мс для 95% запросов. Результаты подтверждают высокую эффективность микросервисного подхода для создания масштабируемых и отказоустойчивых систем в образовательной сфере.

Ключевые слова: микросервисная архитектура, REST API, FastAPI, высокая нагрузка, нагрузочное тестирование, RabbitMQ, Docker, асинхронное программирование, веб-платформа

A. E. Korovyanskiy¹✉, A. A. Sharapov¹

Development and research of the server part of a web platform for managing scientific conferences using REST API

¹Siberian State University of Geosystems and Technologies, Novosibirsk, Russian Federation
e-mail: lexparker925@gmail.com

Abstract. This article addresses the issue of insufficient performance and fault tolerance of traditional monolithic web systems for managing scientific events under peak loads. An approach to developing a server-side based on a microservice architecture is proposed and investigated. The process of decomposing the system into independent services is described, and the choice of a technology stack, including the FastAPI framework, RabbitMQ message broker, and Docker containerization technology, is justified. A key part of the research is the execution and analysis of load testing results using the Locust tool. It has been experimentally proven that the proposed architecture, after optimization, can stably handle loads of up to 250 requests per second with a response time not exceeding 450 ms for 95% of requests. The results confirm the high efficiency of the microservice approach for creating scalable and fault-tolerant systems in the educational sphere.

Keywords: microservice architecture, REST API, FastAPI, high load, load testing, RabbitMQ, Docker, asynchronous programming, web platform

Введение

Специфика администрирования научных конференций сопряжена с рядом критических инфраструктурных и технологических проблем. Основная проблема проведение таких мероприятий заключается в том, что периоды низкой активности сменяются резкими всплесками в дни окончания приема заявок. Резкое повышение нагрузки становится настоящим стресс-тестом для серверной инфраструктуры. Системы, основанные на монолитной архитектуре, могут не справляться с этой задачей.

Анализ существующих решений, показывает, что они, как правило, базируются на универсальных CMS или устаревших монолитных фреймворках. Это приводит к низкой производительности, нестабильной работе и к потере данных и негативному опыту пользователей. Такие системы дороги в поддержке и практически не поддаются гибкому масштабированию. Для обработки пиковой нагрузки приходится увеличивать мощность всего сервера, что экономически неэффективно.

Целью данной работы является разработка системы, а также исследование и практическое доказательство эффективности применения микросервисной архитектуры для создания производительной, отказоустойчивой и масштабируемой серверной платформы для управления научными конференциями.

Методы и материалы

Для достижения поставленной цели был применен комплексный подход, включающий проектирование распределенной системы, выбор современного технологического стека и проведение проверки производительности.

В основе системы лежит микросервисная архитектура [1]. Система разделена на четыре сервиса с разграниченными зонами ответственности. Процессы регистрации, проверки прав доступа и ведения профилей изолированы внутри модуля Auth service. Бизнес-логика, связанная с координацией научных мероприятий, секций и обработкой поступающих заявок, реализована в рамках Core service. Для снижения нагрузки на основной сервис отправка транзакционных писем была вынесена в асинхронный Notification service. Analytics service отвечает за сбор логов и системных метрик.

Взаимодействие между сервисами осуществляется как синхронно, так и асинхронно через брокер сообщений RabbitMQ, что позволяет разгрузить основной API от тяжелых операций [2–4]. Все внешние запросы проходят шлюз API, реализованный на базе Nginx [5]. Он является единой точкой входа для всей системы [6].

Выбор технологий для реализации проекта был продиктован требованиями к высокой производительности, отказоустойчивости и гибкости архитектуры. В качестве основного фреймворка был выбран FastAPI [7–9]. В отличие от Django, данный фреймворк обладает поддержкой асинхронности через библиотеку `asyncio` [10]. Поддержка асинхронности позволяет системе эффективно обрабатывать большой поток одновременных I/O-операций, включая запросы к базе

данных и внешним сервисам. Это является важным фактором для стабильной работы API.

Для реализации асинхронного взаимодействия между компонентами системы был внедрен брокер сообщений RabbitMQ [11, 12]. Данное решение используется для организации очередей задач с гарантированной доставкой сообщений. Использование брокера сообщений позволяет добиться слабой связности сервисов системы. В качестве системы управления базами данных была выбрана реляционная СУБД PostgreSQL [13]. PostgreSQL обладает высокой надежностью, имеет расширенную поддержку сложных типов данных.

Развертывание и масштабирование было обеспечено с помощью технологий контейнеризации Docker и Docker Compose [14]. Docker позволяет упаковать каждый сервис со всеми зависимостями в изолированный контейнер. В свою очередь, инструмент Docker Compose предоставляет возможность управлять сложной инфраструктурой и достаточно быстро запускать всю систему как единое целое.

Для проверки производительности системы был использован фреймворк Locust [15, 16]. Для проверки был разработан сценарий, имитирующий поведение реальных пользователей. По сценарию после аутентификации виртуальные пользователи в случайном порядке выполняли запросы к API, преимущественно запрашивая списки своих заявок [17].

Нагрузочное тестирование проводилось в два этапа. На первом этапе было проведено моделирование 100 одновременных пользователей для проверки соответствия базовым требованиям. На втором этапе было количество пользователей было увеличено до 300 для выявления узких мест и исследования пределов масштабируемости системы.

Результаты

В результате разработки была создана полнофункциональная серверная часть, состоящая из шести Docker-контейнеров, управляемых через Docker Compose. Реализованные программные модули обеспечивают корректное выполнение всех базовых операций, начиная от авторизации пользователей и заканчивая асинхронной отправкой сервисных уведомлений. Данная система была развернута на сервере университета.

Замеры производительности при умеренной нагрузке с помощью инструмента Postman [18] показали низкое время отклика для ключевых эндпоинтов, в среднем 13.8 мс, что превосходит заявленное в требованиях технического задания (менее 300 мс). Результаты первого этапа нагрузочного тестирования показаны в табл. 1.

Таблица 1. Результаты теста при нагрузке в 100 пользователей

Метрика	Значение
Пиковая производительность (RPS)	85 запросов/сек
Время отклика (95-й перцентиль)	240
Количество ошибок	0

На втором этапе нагрузочного тестирования была выявлена деградация производительности, связанная с исходной конфигурацией запуска FastAPI-сервисов в одном рабочем процессе. Данный дефект был устранен путем перехода на продакшен-конфигурацию с использованием Gunicorn [19] в качестве менеджера процессов. В итоге конфигурация была усилена путем добавления четырех асинхронных Unicorn-воркеров для каждого сервиса [20]. Результаты тестирования показаны в табл. 2. Графики продемонстрированы на рис. 1.

Таблица 2. Результаты теста при нагрузке в 300 пользователей

Метрика	Значение
Пиковая производительность (RPS)	250 запросов/сек
Время отклика (95-й перцентиль)	450
Количество ошибок	0

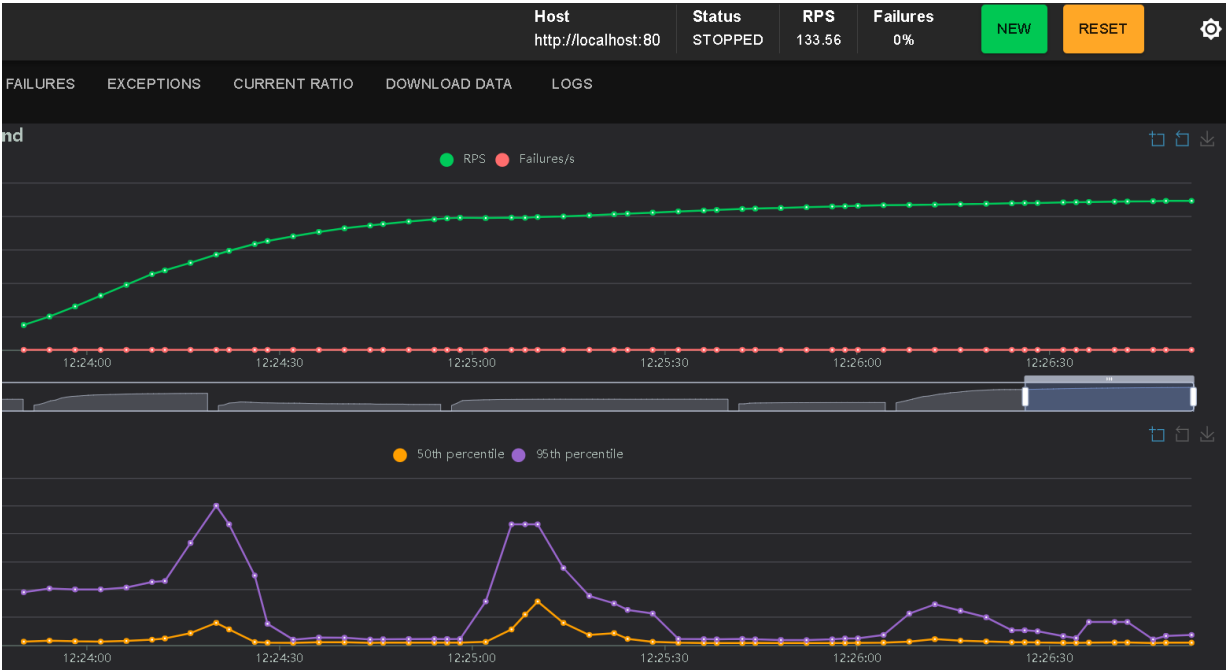


Рис. 1. Результаты нагрузочного тестирования

Обсуждение

Полученные результаты наглядно демонстрируют преимущества выбранного архитектурного подхода. Низкое время отклика при единичных запросах подтверждает высокую эффективность фреймворка FastAPI и асинхронного драйвера базы данных.

Наиболее важным результатом исследования является реакция системы на стресс-тест. Первоначальная деградация производительности и ее последующее успешное устранение путем горизонтального масштабирования на уровне процессов доказывают два ключевых положения. Во-первых, система ведет себя предсказуемо под нагрузкой. Во-вторых, микросервисная архитектура обеспечивает необходимую гибкость для точечной оптимизации производительности, что

было бы крайне затруднительно в монолитной системе, где пришлось бы масштабировать все приложение целиком.

Стабильная работа на уровне 250 запросов в секунду без ошибок подтверждает, что система обладает значительным запасом прочности и способна выдерживать пиковые нагрузки, многократно превышающие типовые для подобных проектов.

Заключение

В результате была успешно решена задача разработки и исследования серверной части веб-платформы для управления научными конференциями. По результатам тестирования можно сделать вывод, что применение микросервисной архитектуры в сочетании с современным асинхронным стеком технологий является эффективным решением для создания высокопроизводительных, отказоустойчивых и легко масштабируемых систем.

Данные, полученные в ходе нагрузочного тестирования, подтверждают преимущества микросервисного подхода. Разработанный программный продукт является готовым к внедрению, обладает значительным потенциалом для дальнейшего развития и масштабирования.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ньюмен С. Создание микросервисов. – Санкт-Петербург: Питер, 2016. – 304 с.
2. Ричардсон К. Микросервисы. Паттерны разработки и рефакторинга. – Санкт-Петербург: Питер, 2019. – 544 с.
3. Филдинг, Р. Т. Архитектурные стили и проектирование сетевых программных архитектур : диссертация на соискание степени доктора философии / Рой Томас Филдинг ; перевод с английского. – Ирвайн : Калифорнийский университет, 2000.
4. Фаулер М. Микросервисная архитектура //martinfowler.com. – 2014. – URL: <https://martinfowler.com/articles/microservices.html> (дата обращения: 15.05.2026).
5. Nginx: официальная документация / Nginx, Inc. – 2026. – Текст : электронный. – URL: <https://nginx.org/ru/docs> (дата обращения: 25.05.2026).
6. Ричардсон, Л. RESTful веб-API : [перевод с английского] / Леонард Ричардсон, Майк Амундсен. – Москва : О'Рейли, 2014. – 402 с. – Текст : непосредственный.
7. FastAPI: официальная документация. – URL: <https://fastapi.tiangolo.com/> (дата обращения: 20.05.2026).
8. Pydantic: официальная документация / S. Colvin. – 2024. – Текст : электронный. – URL: <https://docs.pydantic.dev/latest/> (дата обращения: 09.05.2026).
9. Рамальо, Л. Python. К вершинам мастерства : [перевод с английского] / Лучано Рамальо. – Москва : ДМК Пресс, 2021. – 768 с. – Текст : непосредственный.
10. asyncio – Asynchronous I/O : официальная документация Python 3.10 / Python Software Foundation. – 2024. – Текст : электронный. – URL: <https://docs.python.org/3.10/library/asyncio.html> (дата обращения: 11.05.2024).
11. Видник, А. RabbitMQ в действии. Распределенные приложения на основе очередей сообщений : [перевод с английского] / Альваро Видник, Джейсон Уильямс. – Москва : ДМК Пресс, 2014. – 354 с. – Текст : непосредственный.
12. RabbitMQ: официальная документация / Broadcom. – 2026. – Текст : электронный. – URL: rabbitmq.com (дата обращения: 25.05.2026). Nginx: официальная документация / F5, Inc. – 2024. – Текст : электронный. – URL: <http://nginx.org/ru/docs/> (дата обращения: 10.05.2024).

13. PostgreSQL 15: официальная документация / The PostgreSQL Global Development Group. – 2024. – Текст : электронный. – URL: <https://www.postgresql.org/docs/15/index.html> (дата обращения: 10.05.2024).
14. Тернбулл, Д. Docker: разработка и эксплуатация : [перевод с английского] / Джеймс Тернбулл, Карл Мэттиас. – Санкт-Петербург : Питер, 2017. – 336 с. – Текст : непосредственный.
15. Locust: официальная документация. – URL: <https://docs.locust.io/> (дата обращения: 25.05.2024).
16. Locust. Running Locust distributed: официальная документация / Locust Project. – 2026. – Текст : электронный. – URL: locust.io (дата обращения: 25.05.2026)
17. Бек, К. Экстремальное программирование. Разработка через тестирование : [перевод с английского] / Кент Бек. – Санкт-Петербург : Питер, 2003. – 224 с. – Текст : непосредственный.
18. Postman Learning Center: официальная документация / Postman, Inc. – 2026. – Текст : электронный. – URL: <https://learning.postman.com/docs> (дата обращения: 25.05.2026).
19. Gunicorn - Green Unicorn: официальная документация / В. Т. Jones. – 2024. – Текст : электронный. – URL: <https://gunicorn.org/> (дата обращения: 11.05.2024).
20. Uvicorn: официальная документация / Encode OSS Ltd. – 2026. – Текст : электронный. – URL: <https://uvicorn.dev> (дата обращения: 25.05.2026).

© А. Е. Коровянский, А. А. Шаранов, 2026