DOI: 10.58168/PHBSc-IT2025_222-229
УДК 004.78

РАЗРАБОТКА АРХИТЕКТУРЫ ГОРИЗОНТАЛЬНОГО МАСШТАБИРОВАНИЯ В ВЕБ-СЕРВИСЕ ИНТЕРАКТИВНОЙ ОНЛАЙН-ДОСКИ

THE DEVELOPMENT OF A HORIZONTAL SCALING ARCHITECTURE FOR A WEB-BASED INTERACTIVE ONLINE WHITEBOARD SERVICE

Горбунов В.А., доктор физико-математических наук, профессор, профессор, ФГБОУ ВО «Вологодский государственный университет», Вологда, Россия

Gorbunov V.F., DrSc in Physics and Mathematics, professor, professor, Vologda State University, Vologda, Russia

Кочкин Д.В., кандидат технических наук, доцент, доцент, ФГБОУ ВО «Вологодский государственный университет», Вологда, Россия

Kochkin D.V., PhD in technical science, Docent, Associate Professor, Vologda State University, Vologda, Russia

Самойлов М.А., студент 4-го курса Института Математики, Естественных и Компьютерных Наук, ФГБОУ ВО «Вологодский государственный университет», Вологда, Россия

Samoylov M.A., 4th year student of the Institute of Mathematics, Natural and Computer Sciences, FSBEI HE «Vologda State University», Vologda, Russia

Аннотация. В данной статье рассматривается проблема проектирования архитектуры горизонтального масштабирования для веб-сервиса интерактивной онлайн-доски. Разработка включает использование современных подходов и технологий, таких как микросервисная модель, Spring Boot Framework, WebSocket, RabbitMQ, Redis, Docker, Nginx и MongoDB. Целью работы является создание масштабируемого и отказоустойчивого решения. В статье анализируются преимущества и недостатки выбранных технологий, а также их вклад в обеспечение высокой производительности и надёжности веб-сервиса. Полученные результаты могут быть полезны для разработки аналогичных систем, требующих высокой степени автоматизации и масштабируемости.

Ключевые слова: микросервисы, RabbitMQ, REST API, Docker, балансировка нагрузки, горизонтальное масштабирование.

Abstract. The present paper sets out the problem of designing a horizontal scaling architecture for a web-based interactive online whiteboard service. The development employs contemporary approaches and technologies, including the microservice model, the Spring Boot Framework, WebSocket, RabbitMQ, Redis, Docker, Nginx and MongoDB. The objective of the paper is to create a scalable and fault-tolerant solution. The study methodically evaluates the merits and drawbacks of the chosen technologies, and their collective impact on the web service's performance and reliability. The results obtained can be useful for the development of similar systems that require a high degree of automation and scalability.

Keywords: microservices, RabbitMQ, REST API, Docker, load balancing, horizontal scaling.

Введение

Рост популярности удалённой работы и обучения требует инструментов для совместной работы в реальном времени. Интерактивные онлайн-доски, такие как Miro или Jamboard, обеспечивают визуальную коммуникацию, однако их архитектура часто не адаптирована для специфических требований, таких как высокая нагрузка или интеграция с внутренними системами. Основная проблема заключается в обеспечении горизонтального масштабирования, позволяющего справляться с увеличением числа пользователей без потери производительности [1, 2].

Целью данной работы является разработка архитектуры веб-сервиса интерактивной онлайн-доски, способной масштабироваться горизонтально. Для этого предложено сочетание микросервисной модели, брокера сообщений RabbitMQ, распределённого кэша Redis и балансировщика нагрузки Nginx.

Постановка задачи

Основная задача в разработке архитектуры горизонтального масштабирования заключается в обеспечении бесперебойной работы пользователей в реальном времени даже при больших нагрузках.

Для решения данной задачи необходимо:

1. Разделить систему на независимые модули (клиент, сервер, кэш, брокер сообщений, база данных).

2. Использовать WebSocket для двусторонней связи в реальном времени между клиентом и сервером.

3. Применить Docker для развёртывания реплик серверной части.

4. Интегрировать балансировщик нагрузки Nginx.

Архитектура системы.

Система состоит из нескольких модулей, каждый из которых выполняет определённые функции и взаимодействует с другими модулями через чётко определённые интерфейсы. Ниже приведены описание основных модулей и схема их взаимосвязи (рисунок 1):

Frontend – реализован на React и Excalidraw, обеспечивает взаимодействие с пользователем.

Backend – микросервисы на Spring Boot, обрабатывающие запросы через REST API и WebSocket.

Nginx – балансировщик нагрузки, распределяющий запросы между репликами Backend.

Redis – распределённое хранилище для кэширования изменений на досках.

MongoDB – основная база данных для хранения информации о пользователях и досках.

RabbitMQ – брокер сообщений для синхронизации данных между репликами.

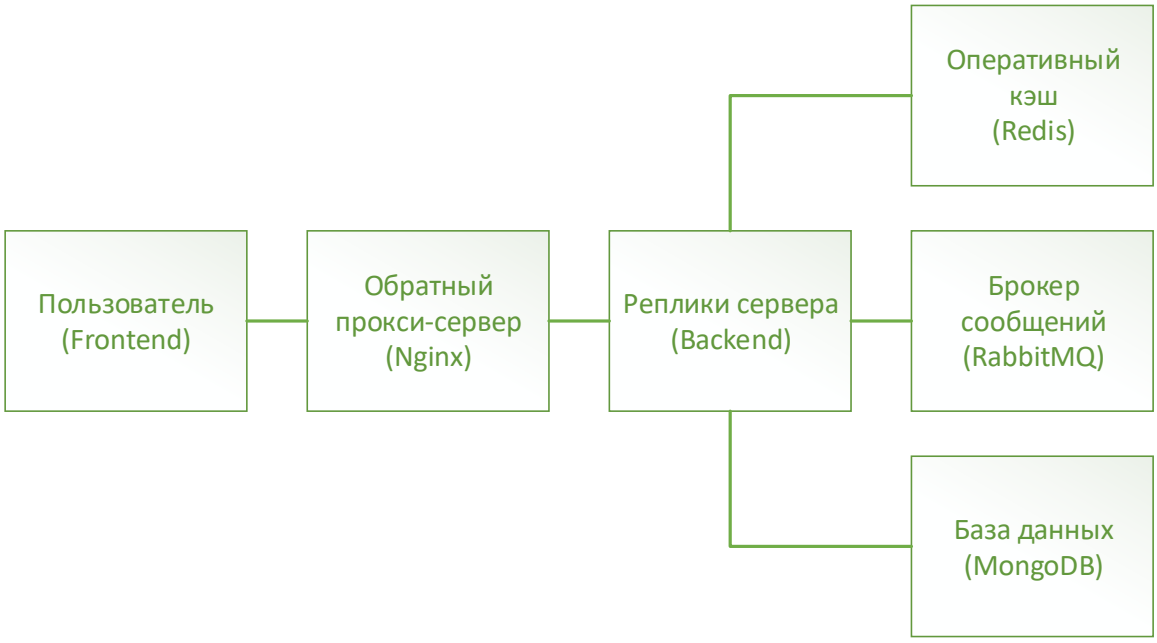

Рисунок 1 – Схема взаимосвязей модулей

Для масштабирования модуль Backend развёртывается в виде нескольких реплик, управляемых Docker. Конфигурация Docker Compose (Табл. 1) включает:

1) 3 реплики Backend;

2) MongoDB, Redis и RabbitMQ в отдельных контейнерах;

3) Nginx для распределения запросов.

Таблица 1 – Конфигурация Docker Compose

Сервис	Образ	Порт	Назначение
Backend	backend-app	8080:8080	Обработка запросов
Nginx	nginx:latest	5173:80	Балансировка нагрузки
RabbitMQ	rabbitmq:3-management	5672:5672	Обмен сообщениями
Redis	redis:latest	6379:6379	Кэширование данных
MongoDB	mongo:latest	27017:27017	Хранение данных

Разработанная архитектура базируется на комбинации современных шаблонов проектирования и технологий, обеспечивающих масштабируемость, отказоустойчивость и высокую производительность системы.

1. Микросервисная архитектура

Система разделена на независимые компоненты: Frontend, Backend, Redis, RabbitMQ и MongoDB. Каждый модуль развёртывается и масштабируется автономно [3, 4], что позволяет: оптимизировать ресурсы. Например, увеличить количество реплик Backend при росте нагрузки, не затрагивая другие компоненты; упростить поддержку, так как обновление одного сервиса не требует остановки всей системы; повысить отказоустойчивость, поскольку сбой в одном модуле не приводит к остановке работы остальных.

Единственным недостатком данной архитектуры является накладные расходы, вызванные обменом данными между сервисами через REST API.

2. Паттерн «Издатель-Подписчик»

Для синхронизации изменений между репликами Backend используется брокер сообщений RabbitMQ. Принцип работы заключается в определении издателя (активная реплика Backend), который отправляет событие об изменении в очередь RabbitMQ, и подписчиков (все реплики), которые получают сообщение и обновляют локальный кэш в Redis. Это обеспечивает: асинхронную обработку, что снижает задержки при высокой нагрузке; согласованность данных, благодаря чему все реплики получают и обеспечивают актуальное состояние доски.

Выбор RabbitMQ обусловлен поддержкой Spring Boot, высокой пропускной способностью и надёжностью доставки сообщений [5, 6].

Из-за отсутствия настройки подтверждённой доставки имеется риск потери сообщений, однако в применяемой системе важнее мгновенная синхронизация изменений, чем гарантированность доставки сообщений.

3. Распределённый кэш Redis

Redis используется для хранения текущих изменений на досках в памяти, что позволяет: снизить нагрузку на MongoDB, так как частые операции чтения и записи выполняются через Redis, а в MongoDB данные компонуются и записываются через определённые интервалы времени; ускорить отклик, так как Redis является резидентной СУБД.

Кэш организован по принципу ключ-значение, где ключом является идентификатор доски, а значением – сериализованный JSON с изменениями.

Следствием использования резидентной базы данных является риск потери части данных вследствие аварии. Однако риски обоснованы снижением задержки при операции чтения и нагрузки на основную базу данных при записи, также они минимизируются благодаря короткому интервалу времени между переносом данных.

4. Балансировка нагрузки через Nginx

Nginx выполняет две ключевые роли: распределение HTTP-запросов: Используется алгоритм Round Robin для равномерного распределения нагрузки между репликами Backend; управление WebSocket-соединениями: Nginx перенаправляет долгоживущие подключения на одну реплику, чтобы избежать разрывов сессий.

Конфигурация Nginx включает оптимизацию буферов и таймаутов, что повышает стабильность при пиковых нагрузках.

Несмотря на преимущества, балансировщик нагрузки является единой точкой отказа системы, поэтому требует глубокого понимания сетевых процессов для обеспечения его работоспособности.

5. Шаблон «Репозиторий»

Для работы с MongoDB применён паттерн «Репозиторий», который абстрагирует доступ к данным: репозитории инкапсулируют CRUD-операции, а сервисный слой использует их для бизнес-логики, что упрощает тестирование и замену источника данных.

Из недостатков применения данного шаблона можно отнести возможную избыточность кода и сложность реализации комплексных запросов. Однако этот подход обеспечивает изоляцию бизнес-логики, и, как следствие, гибкость, поскольку переход на другую СУБД потребует изменения только репозиториев, а не всей системы [7].

6. Внедрение зависимости

Spring Boot Framework автоматически управляет зависимостями между компонентами: через контекст приложения создаются и связываются бины (объекты, управляемые Spring) [8]. К преимуществам данного подхода можно отнести упрощение замены зависимостей. Например, настоящие объекты заменяются на mock-объекты в unit-тестах.

Данная технология может вызывать сложность настройки при большом количестве связей и скрывать зависимости, однако её применение обусловлено сокращением времени разработки за счёт использования автоматической конфигурации.

7. Связь клиента и сервера через WebSocket

WebSocket используется для обеспечения двусторонней связи между клиентом и сервером, что критически важно для мгновенной синхронизации изменений на интерактивной доске. Принцип работы включает установление постоянного соединения без повторных HTTP-запросов и распространение событий при различных изменениях.

Преимуществом данного протокола является низкая задержка даже при частых обновлениях, обусловленная отсутствием накладных расходов на установку соединения для каждого события. К недостатку WebSocket можно отнести высокое потребление ресурсов (каждое соединение требует долговременного выделения памяти). Данная технология используется из-за эффективности работы в реальном времени [9].

Тестирование

Для оценки производительности системы проведено нагрузочное тестирование с использованием Apache JMeter. Тестирование имитировало сценарий одновременной работы пользователей, выполняющих операции рисования, добавления элементов и синхронизации изменений [10, 11]. Результаты представлены в таблице 2.

Таблица 2 – Результаты нагрузочного тестирования

Конфигурация	Среднее время ответа сервера (мс)	Доля ошибок в обработке запросов (%)	Среднее количество отправляемых запросов в секунду
Одна реплика backend	6714	4,59	408
Три реплики backend	7234	0	612

Анализ данных показывает, что увеличение числа реплик Backend с 1 до 3 снизило долю ошибок до нуля за счёт распределения нагрузки, однако среднее время ответа возросло на 28%. Это связано с накладными расходами на синхронизацию данных между репликами через RabbitMQ и Redis. При этом пропускная способность системы (количество запросов в секунду) увеличилась на 50%, что подтверждает эффективность горизонтального масштабирования.

Важно отметить, что тестирование проводилось на локальном компьютере, где ресурсы процессора и памяти были разделены между сервисами приложения и инструментом тестирования. Это привело к конкуренции за ресурсы и ограничило многопоточную обработку. В промышленном развёртывании с выделенными серверами для Backend, Redis и RabbitMQ ожидается снижение времени ответа и рост количества обрабатываемых запросов.

Таким образом, результаты тестирования подтверждают, что архитектура способна масштабироваться горизонтально, обеспечивая отказоустойчивость и стабильную работу при высокой нагрузке.

Результаты

Разработанная архитектура продемонстрировала значительные результаты в увеличении производительности, поддержки масштабируемости и отказоустойчивости. Основные достижения и решённые проблемы приведены ниже.

1. Высокая производительность:

а) Использование Redis для кэширования текущих изменений снизило задержку при операциях чтения/записи.

б) Балансировка нагрузки (Nginx) и горизонтальное масштабирование позволили обрабатывать большее количество запросов в секунду.

2. Масштабируемость:

а) Микросервисная модель и Docker обеспечивают лёгкое добавление реплик Backend.

б) RabbitMQ гарантирует синхронизацию данных между репликами без конфликтов версий.

3. Отказоустойчивость:

а) При падении одной реплики Backend система продолжает работать за счёт автоматического перенаправления запросов через Nginx.

б) Redis минимизирует потерю данных при кратковременных сбоях.

4. Синхронизация в реальном времени:

Комбинация WebSocket и RabbitMQ обеспечивают низкую задержку синхронизации, что критически важно для совместной работы пользователей.

5. Высокая нагрузка на базу данных:

MongoDB становилась узким местом при частых запросах.

Redis взял на себя операции с краткосрочными данными, снизив нагрузку на MongoDB.

6. Рассогласованность данных между репликами:

Изменения на одной реплике не отражались на других.

RabbitMQ (паттерн «Издатель-Подписчик») синхронизировал данные через асинхронные сообщения.

7. Медленный отклик при росте пользователей:

Один сервер Backend не справлялся с нагрузкой.

Горизонтальное масштабирование и Nginx распределили запросы между репликами.

8. Уязвимость к сбоям:

Единый сервер Backend создавал риск полного отказа.

Репликация Backend и Health Checks в Nginx повысили отказоустойчивость.

Заключение

Разработанная архитектура демонстрирует высокую эффективность в условиях горизонтального масштабирования. Сочетание микросервисов, RabbitMQ и Redis, а также различных архитектурных и проектных шаблонов обеспечивает отказоустойчивость и производительность при нагрузке до 1000 элементов на доску. Дальнейшие исследования могут быть направлены на оптимизацию использования памяти в Redis с крупными досками, внедрение алгоритмов сжатия данных для WebSocket для снижения сетевой нагрузки и уменьшение накладных расходов при синхронизации через RabbitMQ за счёт пакетной обработки сообщений.

СПИСОК ЛИТЕРАТУРЫ

1. Фаулер, М. Паттерны корпоративных приложений / М. Фаулер. – Москва: Диалектика-Вильямс, 2020. – 544 с.

2. Кочкин, Д.В. Проектирование и конструирование программного обеспечения : учебное пособие / Д.В. Кочкин, А.Н. Швецов. – Вологда : Вологод, 2023. – 127 с.

3. Ньюмен, С. Создание микросервисов / С. Ньюмен. – Санкт-Петербург: Питер, 2016. – 304 с.

4. Принципы построения самоорганизующихся информационно-телекоммуникационных систем / А.А. Суконщиков, А.Н. Швецов, И.А. Андрианов, Д.В. Кочкин // Вестник Череповецкого государственного университета. – 2021. – № 1(100). – С. 56-67. – DOI 10.23859/1994-0637-2021-1-100-4.

5. Бартель, Й. Начало работы с AMQP и RabbitMQ / Й. Бартель // Журнал разработки программ. – 2009. – URL: https://www.infoq.com/articles/AMQP-RabbitMQ.

6. Интеллектуальные информационно-телекоммуникационные системы / А.Н. Швецов, А.А. Суконщиков, И.А. Андрианов [и др.]. – Вологда : Вологодский государственный университет, 2023. – 127 с.

7. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – Санкт-Петербург: Питер, 2016. – 368 с.

8. Уоллс, К. Spring в действии / К. Уоллс. – Москва: ДМК Пресс, 2022. – 544 с.

9. Wang, V. The Definitive Guide to HTML5 WebSocket / V. Wang, F. Salim, P. Moskovits. – New York: Apress, 2013. – 227 p.

10. Development of a forecasting agent based on a fuzzy neural Petri net for predicting abnormal situations in automation systems / A.A. Sukonschikov, A.N. Shvetsov, I.A. Andrianov [et al.] // AIP Conference Proceedings, Krasnoyarsk, 29-30 апреля 2021 года. – Vol. 2402. – Melville, New York, United States of America: AIP Publishing, 2021. – P. 50025. – DOI 10.1063/5.0071782.

11. Краснов, А.А. Разработка системы тестов для тестирования веб-приложения / А.А. Краснов, Д.В. Кочкин // Вестник Вологодского государственного университета. Серия: Технические науки. – 2024. – № 3(25). – С. 43-48.

REFERENCES

1. Fowler, M. Patterns of enterprise applications / M. Fowler. – Moscow: Dialectics-Williams, 2020. – 544 p.

2. Kochkin, D.V. Software design and construction: a tutorial / D.V. Kochkin, A.N. Shvetsov. – Vologda: Vologod, 2023. – 127 p.

3. Newman, S. Creating microservices / S. Newman. – St. Petersburg : Piter, 2016. – 304 p.

4. Principles of constructing self-organizing information and telecommunication systems / A.A. Sukonshchikov, A.N. Shvetsov, I.A. Andrianov, D.V. Kochkin // Bulletin of Cherepovets State University. – 2021. – No. 1 (100). – P. 56-67. – DOI 10.23859/1994-0637-2021-1-100-4.

5. Bartel, J. Getting Started with AMQP and RabbitMQ / J. Bartel // Journal of Software Development. – 2009. – URL: https://www.infoq.com/articles/AMQP-RabbitMQ.

6. Intelligent Information and Telecommunication Systems / A.N. Shvetsov, A.A. Sukonshchikov, I. A. Andrianov [et al.]. – Vologda: Vologda State University, 2023. – 127 p.

7. Object-Oriented Design Techniques. Design Patterns / E. Gamma, R. Helm, R. Johnson, D. Vlissides. – St. Petersburg: Piter, 2016. – 368 p.

8. Walls, K. Spring in Action / K. Walls. – Moscow: DMK Press, 2022. – 544 p.

9. Wang, V. The Definitive Guide to HTML5 WebSocket / V. Wang, F. Salim, P. Moskovits. – New York: Apress, 2013. – 227 p.

10. Development of a forecasting agent based on a fuzzy neural Petri net for predicting abnormal situations in automation systems / A.A. Sukonschikov, A.N. Shvetsov, I.A. Andrianov [et al.] // AIP Conference Proceedings, Krasnoyarsk, April 29-30, 2021. – Vol. 2402. – Melville, New York, United States of America: AIP Publishing, 2021. – P. 50025. – DOI 10.1063/5.0071782.

11. Krasnov, A.A. Development of a test system for testing a web application / A.A. Krasnov, D.V. Kochkin // Bulletin of the Vologda State University. Series: Technical Sciences. – 2024. – No. 3(25). – P. 43-48.