



Методы межсервисной коммуникации в микросервисной архитектуре

УДК 004.7

С.В. ШЕВЕЛЕВ, доцент кафедры “Сетевые информационные технологии и сервисы” ФГБОУ ВО “Московский технический университет связи и информатики” кандидат технических наук, **К.В. МАКАРОВ**, магистрант 2-го курса факультета “Информационные технологии”

Методы межсервисной коммуникации в микросервисной архитектуре *Methods of interservice communication in microservice architecture*

В статье рассматриваются современные методы межсервисной коммуникации в микросервисной архитектуре программных систем. Анализируются синхронные и асинхронные подходы, их ключевые характеристики, применяемые протоколы и технологии, а также особенности их использования в различных сценариях построения распределенных систем.

This article examines modern methods of interservice communication in microservice architecture of software systems. It analyzes synchronous and asynchronous approaches, their key characteristics, applied protocols and technologies, as well as features of their use in various scenarios of building distributed systems.

Ключевые слова: микросервисная архитектура, межсервисное взаимодействие, брокеры сообщений, Apache Kafka, RabbitMQ, протокол AMQP, REST-взаимодействие, gRPC.

Keywords: *microservice architecture, inter-service communication, message brokers, Apache Kafka, RabbitMQ, AMQP protocol, REST communication, gRPC.*

Введение

Микросервисная архитектура представляет собой современный архитектурный подход к разработке программных систем, при котором приложение структурируется как набор слабосвязанных, независимо развертываемых сервисов [1]. Каждый микросервис реализует конкретную бизнес-функцию и может разрабатываться, тестироваться и масштабироваться независимо от других компонентов системы. Актуальность методов межсервисной коммуникации обусловлена широким распространением микросервисных архитектур в промышленной разработке: согласно исследованиям, около 85 процентов крупных технологических компаний используют микросервисы в качестве основного архитектурного стиля [2].

Переход от монолитных систем к микросервисной архитектуре создает новые вызовы в организации взаимодействия между компонентами. В монолитных приложениях взаимодействие осуществляется посредством вызовов функций в

рамках единого процесса, что обеспечивает низкую задержку и гарантированную консистенцию данных [3]. В микросервисных системах коммуникация происходит через сеть, что вносит дополнительные задержки, риск отказов и усложняет обеспечение согласованности данных [4]. Кроме того, в распределенных системах возникают новые проблемы: управление сетевыми ошибками, асинхронность операций, необходимость обработки частичных отказов и поддержание консистенции распределенного состояния.

Выбор метода межсервисной коммуникации обладает непосредственной экономической значимостью. Согласно исследованиям, производительность систем критична для бизнес-результатов: компания Amazon обнаружила, что каждые 100 миллисекунд увеличения задержки обходятся ей в один процент потерь продаж [5]. Для современного объема операций Amazon это означает потерю порядка нескольких миллиардов долларов в год. Исследование Akamai показало, что

даже 100-миллисекундная задержка загрузки веб-сайта снижает коэффициент конверсии на семь процентов [6], а для систем электронной коммерции каждая дополнительная секунда задержки приводит к потере 7 — 11 процентов конверсии в зависимости от отрасли [30]. Эти цифры демонстрируют, что оптимизация межсервисной коммуникации — это не технический вопрос, а стратегический фактор конкурентоспособности.

Экспериментальные исследования показывают, что выбор протокола коммуникации оказывает прямое влияние на производительность системы, увеличивая или уменьшая ее в несколько раз [7]. Например, технология gRPC демонстрирует повышение производительности на 55,9 — 88,9 процента по сравнению с традиционными HTTP-подходами, в то время как Apache Kafka может обрабатывать до миллиона сообщений в секунду против 4000 — 10 000 у RabbitMQ [8].

**Статью целиком читайте
в бумажной версии журнала**