Материалы XXXI Международной научной конференции
«ОБЩЕСТВО: НАУЧНО-ОБРАЗОВАТЕЛЬНЫЙ ПОТЕНЦИАЛ РАЗВИТИЯ (идеи, ресурсы, решения)»
(г. Москва, Россия, 25 июня 2025 г.)

КОМПЬЮТЕРНЫЕ НАУКИ И ИНФОРМАТИКА

МИКРОСЕРВИСНАЯ АРХИТЕКТУРА: АНАЛИЗ СОВРЕМЕННЫХ ПОДХОДОВ

КОЛОБОВА Дарья Алексеевна
АПРЕЛЕВА Марианна Александровна
Научный руководитель: **КОБЫЛКИН Дмитрий Сергеевич**
кандидат технических наук, доцент
Оренбургский государственный университет
г. Оренбург, Россия

В статье исследуется микросервисная архитектура (MSA) как современный подход к проектированию систем. Рассмотрены ключевые аспекты: декомпозиция, контейнеризация, безопасность и мониторинг. Анализируются перспективы развития MSA.
Ключевые слова: микросервисная архитектура, распределенные системы, контейнеризация, оркестрация сервисов, согласованность данных, межсервисное взаимодействие, масштабируемость.

Современные ИТ-системы требуют высокой гибкости и масштабируемости, что привело к переходу от монолитных архитектур к микросервисным (MSA). Этот подход революционизирует проектирование сложных систем. В статье рассматриваются ключевые аспекты MSA – от базовых принципов до передовых практик внедрения [2].

Микросервисная архитектура (MSA) основана на разделении системы на независимые слабосвязанные сервисы, каждый из которых выполняет конкретную бизнес-функцию. Например, в e-commerce отдельные микросервисы могут отвечать за каталог товаров, корзину покупок и платежи. Такой подход обеспечивает независимость разработки и развертывания компонентов, что значительно ускоряет процесс внедрения новых функций и обновлений.

Одним из ключевых преимуществ MSA является возможность использования разных технологических стеков для различных сервисов [1]. В то время как один сервис может быть написан на Java с использованием Spring Boot, другой может использовать Node.js, а третий - Python с Django. Это позволяет выбирать оптимальный инструмент для каждой конкретной задачи и постепенно модернизировать систему, не переписывая ее полностью. Однако такая гибкость требует тщательного планирования интерфейсов взаимодействия между сервисами.

Для эффективной работы микросервисной архитектуры необходима соответствующая инфраструктура. Контейнеризация с помощью Docker стала фактическим стандартом для упаковки микросервисов. Контейнеры обеспечивают изоляцию сервисов и единообразие

среды выполнения, что значительно упрощает развертывание. Оркестратор Kubernetes, в свою очередь, решает задачи управления кластером контейнеров, обеспечивая автоматическое масштабирование, балансировку нагрузки и самовосстановление системы.

Особого внимания заслуживает вопрос организации взаимодействия между микросервисами. Существует два основных подхода: синхронное взаимодействие через REST API и асинхронное взаимодействие через брокеры сообщений (Kafka, RabbitMQ). REST API проще в реализации и отладке, но создает жесткие зависимости между сервисами. Асинхронная модель на основе событий обеспечивает большую гибкость и отказоустойчивость, но требует более сложной архитектуры и дополнительных механизмов обеспечения согласованности данных.

Одной из наиболее сложных задач при работе с MSA является управление распределенными данными. В отличие от монолита, где обычно используется единая база данных, в микросервисной архитектуре каждый сервис имеет свою собственную БД. Это обеспечивает независимость сервисов, но создает проблему поддержания согласованности данных. Для решения этой проблемы применяются различные паттерны, такие как Saga, которая разбивает распределенную транзакцию на последовательность локальных транзакций с компенсирующими действиями [3], или CQRS (Command Query Responsibility Segregation), разделяющий операции записи и чтения данных.

Мониторинг и отладка распределенной системы – еще один серьезный вызов. Традиционные подходы к мониторингу, разработанные для монолитных систем, часто оказываются неэффективными в микросервисной среде. Современные решения, такие как распределенная трассировка (Jaeger, Zipkin), централизованный сбор логов (ELK Stack) и метрик (Prometheus с Grafana), позволяют получить полную картину работы системы. Особенно важна реализация сквозной идентификации запросов (correlation ID), которая помогает отслеживать путь запроса через множество сервисов.

Безопасность в микросервисной архитектуре требует особого подхода. Традиционные механизмы аутентификации и авторизации, основанные на сессиях, плохо подходят для распределенных систем. Вместо них используются токен-ориентированные подходы, такие как JWT (JSON Web Tokens), которые могут передаваться между сервисами и содержать всю необходимую информацию о правах доступа. Service Mesh (например, Istio) предоставляет дополнительные возможности для обеспечения безопасности, такие как автоматическое шифрование трафика между сервисами и детализированное управление политиками доступа.

Несмотря на все преимущества, микросервисная архитектура подходит не для всех проектов. Переход на MSA оправдан, когда система становится достаточно сложной, а команда – достаточно большой [4]. Для стартапов и небольших проектов монолитная архитектура часто остается более предпочтительным выбором, так как требует меньших накладных расходов на инфраструктуру и управление. Однако по мере роста системы и команды преимущества MSA становятся все более очевидными.

Проведенный анализ позволяет заключить, что микросервисная архитектура (MSA) является закономерным этапом эволюции распределенных систем, обеспечивая масштабируемость и отказоустойчивость. Однако ее эффективность напрямую зависит от уровня технологической инфраструктуры и квалификации разработчиков. Наибольшую выгоду MSA приносит в высоконагруженных системах со сложной бизнес-логикой. Перспективными направлениями развития являются гибридные архитектуры и совершенствование инструментов оркестрации.

СПИСОК ЛИТЕРАТУРЫ

1. *Гамма Э. и др.* Приемы объектно-ориентированного проектирования. Паттерны проектирования / пер. с англ. – СПб.: Питер, 2020. – 496 с.

2.*Колобов А.Н.* Информационно-образовательные технологии // Университетский комплекс как региональный центр образования, науки и культуры: сб. материалов Всерос. науч.-метод. конф., – Оренбург, 2023. – С. 1309-1315.

3. *Ньюмен С.* Создание микросервисов / пер. с англ. – М.: ДМК Пресс, 2016. – 280 с.
4. *Ричардсон К.* Микросервисы. Паттерны разработки и рефакторинга / пер. с англ. – М.: Вильямс, 2019. – 432 с.

MICROSERVICE ARCHITECTURE: AN ANALYSIS OF MODERN APPROACHES

KOLOBOVA Darya Alekseevna
APRELEVA Marianna Alexandrovna
Scientific Supervisor: **KOBYLKIN Dmitry Sergeevich**
Candidate of Sciences in Technology, Associate Professor
Orenburg State University
Orenburg, Russia

The article explores microservice architecture (MSA) as a modern approach to system design. The key aspects are considered: decomposition, containerization, security and monitoring. The prospects for the development of MSA are analyzed.
Keywords: microservice architecture, distributed systems, containerization, service orchestration, data consistency, inter-service interaction, scalability.

ОПТИМИЗАЦИЯ SQL-ЗАПРОСОВ: ИНДЕКСЫ, АНАЛИЗ ПЛАНОВ ВЫПОЛНЕНИЯ И ДЕНОРМАЛИЗАЦИЯ

КОЛОБОВА Дарья Алексеевна
ДРАГУН Виктория Александровна
Научный руководитель: **КОБЫЛКИН Дмитрий Сергеевич**
кандидат технических наук, доцент
Оренбургский государственный университет
г. Оренбург, Россия

Целью данной статьи является анализ методов оптимизации SQL-запросов, включая применение индексов, их влияние на производительность запросов, денормализацию данных и анализ планов выполнения с помощью команды EXPLAIN.
Ключевые слова: PostgreSQL, оптимизация запросов, индексы, EXPLAIN, денормализация, производительность БД.

В условиях роста объемов данных и увеличения нагрузки на базы данных эффективность выполнения SQL-запросов становится критически важной. Оптимизация SQL-запросов представляет собой комплекс мер, направленных на повышение эффективности выполнения запросов к базе данных. Она позволяет сократить время отклика, уменьшить нагрузку на сервер и обеспечить стабильную работу приложения даже при больших объемах данных [3].

СУБД PostgreSQL предлагает широкий набор инструментов для анализа и улучшения производительности запросов. Основные методы оптимизации SQL-запросов включают использование индексов, анализ планов