**Проектирование микросервисной архитектуры**

**Докладчики:**

Усов Роман Романович, Костяков Никита Андреевич, Самарин Дмитрий Васильевич

**Группа:** 4134к

# Введение

Сегодня мы представляем вам исследование по **проектированию микросервисной архитектуры**, основанное на нашем опыте разработки **онлайн-сервиса для торгов**.

<https://torgi.monety.shop/>

Наша работа была посвящена созданию онлайн-платформы для проведения аукционов, в рамках которой пользователи могут регистрироваться, делать ставки и получать уведомления о торгах в реальном времени. Первоначально этот сервис был спроектирован в монолитной архитектуре, однако при детальном анализе стало очевидно, что переход к микросервисному подходу мог бы значительно улучшить масштабируемость, отказоустойчивость и гибкость системы.

Сегодня мы расскажем вам о:

1. Принципах микросервисной архитектуры и её преимуществах.
2. Сравнении с монолитным подходом на примере моей системы.
3. Реальных примерах влияния микросервисов на эффективность ведущих IT-компаний.

# Что такое микросервисная архитектура?

Микросервисная архитектура – это один из наиболее популярных и гибких подходов к проектированию программных систем. Она основана на разделении приложения на независимые сервисы, которые взаимодействуют друг с другом через API или системы обмена сообщениями. Каждый такой сервис выполняет отдельную бизнес-функцию, что делает систему более отказоустойчивой, масштабируемой и удобной в разработке.

# Краткая история возникновения микросервисов

Хотя концепция разбиения системы на модули существовала давно, микросервисный подход в его современном виде начал активно развиваться в 2010-х годах. До этого подавляющее большинство программных решений строились на монолитной архитектуре, где все модули и компоненты работали как единое целое.

С ростом IT-индустрии возникли проблемы:

1. Обновление больших приложений стало слишком сложным.
2. Изменение одной части системы могло повлиять на всю систему.
3. Масштабирование было дорогостоящим и неэффективным.

Такие гиганты, как Netflix, Amazon, Uber, Google, eBay, столкнулись с этими ограничениями и начали внедрять концепцию микросервисов, позволяющую разрабатывать, развертывать и масштабировать отдельные части системы независимо друг от друга.

# Ключевые принципы микросервисной архитектуры

1. Разделение бизнес-логики на независимые сервисы

В микросервисной архитектуре каждая часть системы выполняет одну конкретную задачу. Например, в интернет-магазине могут быть отдельные микросервисы для:

1. Обработки заказов
2. Регистрации и авторизации пользователей
3. Системы платежей
4. Службы доставки
5. Уведомлений и email-рассылок

Эти микросервисы могут работать независимо друг от друга, что упрощает модификацию, тестирование и развертывание.

2. Независимое развертывание и масштабирование

1. Если один микросервис перегружен, можно масштабировать только его, не затрагивая остальные.
2. Если требуется внести изменения в один модуль – можно обновить только этот микросервис без остановки всей системы.

3. Гибкость в выборе технологий

Каждый микросервис может использовать разные языки программирования, базы данных и инструменты, подходящие для своей задачи. Например:

1. Go – для высоконагруженных API
2. Python – для машинного обучения
3. Node.js – для работы с реальным временем
4. PostgreSQL – для транзакционных данных
5. MongoDB – для хранения логов и событий

Такой подход делает систему более оптимизированной и производительной.

4. Взаимодействие сервисов через API или системы очередей сообщений

Микросервисы могут взаимодействовать друг с другом с помощью:

1. REST API (HTTP-запросы, JSON)
2. GraphQL (гибкие запросы к данным)
3. gRPC (более быстрый двоичный протокол)
4. Системы очередей сообщений (RabbitMQ, Kafka, NATS)

Это позволяет сервисам не зависеть друг от друга напрямую, а взаимодействовать асинхронно.

5. Высокая отказоустойчивость

Если один микросервис выходит из строя, это не ломает всю систему. Другие сервисы могут продолжать работать в штатном режиме.

Например, если сервис платежей недоступен, пользователь все равно может:

1. Добавить товар в корзину
2. Получить уведомления о скидках
3. Просматривать каталог

Таким образом, отказоустойчивость системы повышается, а критические сбои становятся менее разрушительными.

# Технические аспекты микросервисной архитектуры

1. Контейнеризация (Docker, Kubernetes)

Для эффективного управления микросервисами каждый сервис разворачивается в отдельном контейнере. Это позволяет:

1. Запускать микросервисы в изолированной среде.
2. Быстро разворачивать новые версии сервисов.
3. Управлять нагрузкой с помощью Kubernetes, который автоматически масштабирует контейнеры при росте трафика.

2. Автоматизация CI/CD (Continuous Integration / Continuous Deployment)

Микросервисные системы активно используют CI/CD-подход, который включает:

1. Автоматическую проверку кода после каждого изменения.
2. Автоматическое развертывание новых версий сервисов.
3. Мониторинг и откат изменений, если что-то пошло не так.

3. Мониторинг и логирование

Из-за большого количества сервисов важно следить за их работой.

Используются:

1. Prometheus и Grafana – для мониторинга метрик.
2. ELK Stack (Elasticsearch, Logstash, Kibana) – для сбора и анализа логов.
3. Jaeger, Zipkin – для трассировки запросов между сервисами.

# Ключевые преимущества микросервисов

1. Гибкость разработки – легко добавлять новые функции и обновлять существующие.
2. Лучшее масштабирование – можно увеличивать только нагруженные компоненты, а не всю систему.
3. Отказоустойчивость – сбой в одном сервисе не ломает всю систему.
4. Ускорение разработки – команды могут работать независимо друг от друга.
5. Использование разных технологий – можно выбрать лучший стек для каждой задачи.

# Когда микросервисная архитектура НЕ подходит?

1. Слишком сложная для небольших проектов

1. Если приложение небольшое, микросервисы создадут ненужную сложность.
2. Например, для обычного веб-сайта монолит будет лучше.

2. Требует сильной команды DevOps

1. Нужно уметь настраивать контейнеризацию, автоматический деплой, мониторинг.
2. Без грамотного DevOps-специалиста внедрение микросервисов будет сложным.

3. Увеличение сетевой нагрузки

1. В монолите компоненты вызывают друг друга напрямую.
2. В микросервисах данные передаются через сеть, что может замедлить работу системы.

Сравнение микросервисной и монолитной архитектуры:

Почему на начальном этапе мой проект реализован как монолит, учитывая команду из двух разработчиков?

Мы уже рассмотрели ключевые аспекты микросервисной архитектуры, её преимущества и недостатки, а также провели сравнительный анализ с монолитным подходом. Теперь важно не просто теоретически сопоставить эти архитектуры, но и учесть реальные условия разработки – в данном случае маленькую команду из трех человек, работающих над проектом.

1. Почему монолит был наилучшим выбором для старта разработки?

При проектировании любого программного решения важно учитывать ресурсы, время и масштабы разработки. Наш проект разрабатывался тремя людьми, что накладывает определенные ограничения:

1. Ограниченные ресурсы – отсутствие отдельной команды DevOps для настройки сложной инфраструктуры.
2. Необходимость быстрого создания работающего прототипа – MVP (Minimal Viable Product) должен был быть готов в разумные сроки.
3. Фокус на разработку бизнес-логики, а не на поддержание распределенной системы. Проще тестировать и отлаживать – вся логика находится в одном коде.

# Главное преимущество монолита на старте:

# Минимальные накладные расходы на инфраструктуру, быстрое развертывание и удобная отладка кода.

Если бы изначально мы выбрали микросервисную архитектуру, это потребовало бы:

1. Разработки отдельных сервисов и API для связи между ними.
2. Настройки системы контейнеризации (Docker + Kubernetes).
3. Настройки CI/CD для каждого микросервиса.
4. Управления мониторингом и логированием множества сервисов.

Для команды из трех человек это бы значительно увеличило сложность и сроки разработки.

# 2. Почему на начальном этапе монолитная архитектура лучше?

Монолит – это быстрый старт и удобство разработки.

На старте разработки важно быстро проверить работоспособность идеи, а не сразу строить масштабируемую архитектуру.

Как реализовано в нашем проекте?

1. Все функции аукциона, пользователей и уведомлений находятся в одном приложении.
2. Разработано как единый FastAPI-сервис, который работает с MongoDB.
3. Разворачивается в одном контейнере Docker, не требуя сложной оркестрации.

Какие ограничения у монолита, если система будет расти?

1. Невозможно масштабировать отдельные компоненты (например, если растёт нагрузка на аукционы).
2. Любое обновление требует перезапуска всей системы.
3. Сложнее работать над разными частями кода параллельно, если разработчиков станет больше.

Вывод: Монолит – это хороший выбор для старта, но он может стать узким местом при росте нагрузки.

# 3. Когда микросервисная архитектура становится оправданной?

Хотя монолит является оптимальным выбором на старте, при дальнейшем росте системы переход к микросервисной архитектуре будет логичным шагом.

1. Если в проекте станет больше разработчиков, они смогут работать над отдельными микросервисами.
2. Если появится большая нагрузка на аукционы, можно будет масштабировать сервис аукционов отдельно.
3. Если потребуется гибкость, микросервисная архитектура позволит обновлять отдельные модули без остановки всей системы.

# Статистика внедрения микросервисной архитектуры и её влияние на отрасль

Микросервисная архитектура (МСА) продолжает набирать популярность в мировой практике разработки программного обеспечения, предлагая гибкость, масштабируемость и ускоренное внедрение изменений.

Мировые тенденции:

1. Рост использования микросервисов: Согласно отчёту O'Reilly за 2020 год, около 68% компаний уже внедрили микросервисную архитектуру в своих проектах.

https://libeldoc.bsuir.by/bitstream/123456789/52022/1/Shirshov\_Transformaciya.pdf?utm\_source=

1. Преимущества для бизнеса: Исследования показывают, что внедрение МСА способствует повышению эффективности и производительности программных систем, что положительно сказывается на бизнес-показателях компаний.

https://moitvivt.ru/ru/journal/article?id=1561&utm\_source=

Российский сегмент:

В России наблюдается активное развитие ИТ-отрасли, что создаёт благоприятные условия для внедрения современных архитектурных подходов, включая микросервисы.

1. Рост ИТ-рынка: В 2023 году объём российского ИТ-рынка достиг 3,06 трлн рублей, что на 12% больше по сравнению с предыдущим годом. Такой рост обусловлен, в том числе, переходом на отечественные решения и активным развитием новых технологий.
2. Импортозамещение и развитие отечественных решений: После ухода ряда западных вендоров российские компании начали активно разрабатывать собственное программное обеспечение. Около 76% крупных и средних российских компаний инициировали создание собственного ПО, что способствует внедрению современных архитектурных подходов, таких как микросервисы.
3. Рынок CRM-систем: В 2023 году объём российского рынка CRM-систем превысил 28 млрд рублей. Компании всё чаще выбирают решения, основанные на микросервисной архитектуре, что позволяет им быстро адаптироваться к изменениям и обеспечивать гибкость бизнес-процессов.

Влияние на компании:

1. Переход на микросервисную архитектуру позволяет компаниям:
2. Ускорить разработку и внедрение новых функций.
3. Обеспечить масштабируемость и отказоустойчивость систем.
4. Сократить время простоя и повысить удовлетворённость пользователей.

# Таблица: Ключевые статистические данные о микросервисной архитектуре

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Категория | Показатель/Аспект | Описание | Пример значения/статистика | + | - |
| Уровень внедрения в мире | К 2020 году около 68% компаний внедрили микросервисы. |  |  |  |  |
| Производительность | Время отклика (Response Time) | Среднее время обработки запроса микросервисом. | 50-200 мс (в зависимости от нагрузки и оптимизации) | Высокая производительность за счет изоляции сервисов. | Задержки из-за межсервисного взаимодействия (сети, API-вызовы). |
| Производительность | Пропускная способность (Throughput) | Количество запросов, обрабатываемых в единицу времени. | 1000-5000 запросов/сек (в зависимости от инфраструктуры) | Легко масштабировать отдельные сервисы под нагрузку. | Сложность балансировки нагрузки между сервисами. |
| Масштабируемость | Количество микросервисов | Общее число микросервисов в системе. | 50-100 микросервисов (в среднем для крупных компаний) | Гибкость в масштабировании отдельных компонентов. | Увеличение сложности управления множеством сервисов. |
| Масштабируемость | Горизонтальное масштабирование | Возможность добавлять новые экземпляры микросервисов для обработки нагрузки. | Увеличение на 10 экземпляров при росте нагрузки на 20% | Быстрое реагирование на изменения нагрузки. | Высокие затраты на инфраструктуру. |
| Надежность | Время безотказной работы (Uptime) | Процент времени, когда система работает без сбоев. | 99,9% (для хорошо спроектированных систем) | Изоляция сбоев: отказ одного сервиса не влияет на всю систему. | Сложность обеспечения отказоустойчивости всех сервисов. |
| Надежность | Частота отказов (Failure Rate) | Количество сбоев на 1000 запросов. | 0,1% (в хорошо настроенных системах) | Легче локализовать и исправить ошибки. | Необходимость мониторинга и обработки сбоев в каждом сервисе. |
| Сложность управления | Количество зависимостей между микросервисами | Число связей между микросервисами. | 200+ зависимостей (в крупных системах) | Модульность и независимость сервисов. | Высокая сложность управления зависимостями и версионированием. |
| Сложность управления | Время развертывания (Deployment Time) | Среднее время, необходимое для развертывания изменений в микросервисе. | 5-10 минут (при использовании CI/CD) | Быстрое внедрение изменений в отдельные сервисы. | Сложность координации развертывания множества сервисов. |
| Затраты | Стоимость инфраструктуры | Затраты на поддержку инфраструктуры микросервисов. | (в зависимости от масштаба) | Оптимизация затрат за счет масштабирования только нужных сервисов. | Высокие начальные затраты на инфраструктуру и инструменты. |
| Затраты | Затраты на разработку и поддержку | Затраты на команду разработчиков и DevOps. | (в зависимости от масштаба) | Ускорение разработки за счет независимых команд. | Необходимость высококвалифицированных специалистов. |
| Гибкость | Поддержка различных технологий | Возможность использовать разные языки программирования и фреймворки. | Node.js, Python, Java, Go и др. | Выбор лучшего инструмента для каждой задачи. | Сложность интеграции разнородных технологий. |
| Безопасность | Уровень безопасности | Защита данных и API в микросервисной архитектуре. | Зависит от реализации (OAuth, JWT, шифрование) | Изоляция сервисов повышает безопасность. | Сложность обеспечения безопасности всех сервисов. |
| Тестирование | Сложность тестирования | Уровень сложности тестирования микросервисов. | Высокая (необходимость тестирования каждого сервиса и их взаимодействия) | Легче тестировать небольшие изолированные компоненты | Требуется больше времени и ресурсов на интеграционное тестирование |

Однако стоит учитывать, что внедрение МСА требует значительных инвестиций в обучение персонала, изменение инфраструктуры и внедрение систем мониторинга и управления микросервисами.

В целом, мировые и российские тенденции показывают, что микросервисная архитектура становится ключевым элементом в стратегии развития ИТ-систем, способствуя повышению эффективности и конкурентоспособности компаний.

# Заключение: Монолит сейчас, микросервисы – в будущем

1. Для команды из 3 человек монолит – логичный выбор, так как он позволяет быстро разрабатывать и развертывать систему.
2. Микросервисная архитектура становится актуальной при росте нагрузки, когда нужно масштабирование и высокая отказоустойчивость.
3. В будущем, при росте системы, можно разделить функционал на независимые сервисы и сделать систему более гибкой.

Главный вывод:

На начальном этапе разработки нашего проекта монолитная архитектура была наилучшим выбором, учитывая маленькую команду и необходимость быстрой разработки. Однако при дальнейшем развитии переход к микросервисному подходу позволит решить проблемы масштабируемости и отказоустойчивости.

Источники и аналитика:

Основные:

1. O’Reilly Microservices Adoption Report (2020, 2023)
2. GitLab State of DevOps Report (2021)
3. Отчёт Минцифры РФ о развитии российского ИТ-сектора (2023)
4. Аналитика РБК и CNews по внедрению микросервисов в России (2023)
5. Исследование Gartner: прогноз по росту микросервисных систем (2022-2025)

Дополнительные:

1. **Nixys** – "Монолит или микросервисы: какую IT-инфраструктуру предпочитают крупные компании в России?" https://nixys.ru/monolit-ili-mikroservisy-kakuju-it-infrastrukturu-predpochitajut-krupnye-kompanii-v-rossii/
2. **ComNews** – "Почему микросервисная архитектура – фундамент современного банковского IT-ландшафта?" https://www.comnews.ru/content/234270/2024-07-15/2024-w29/1013/pochemu-mikroservisnaya-arkhitektura-fundament-sovremennogo-bankovskogo-it-landshafta
3. **Atlassian** – "Микросервисы против монолита: сравнение архитектур" https://www.atlassian.com/ru/microservices/microservices-architecture/microservices-vs-monolith
4. **WorkSolutions** – "Микросервисы vs монолит: разница архитектур и руководство по переходу" https://worksolutions.ru/blog/mikroservisy-vs-monolit-raznicza-arhitektur-i-rukovodstvo-po-perehodu/