06 октября 2022

# Микросервисная архитектура

**Последние годы набирает популярность микросервисная архитектура приложений. В материале разберемся в основных плюсах и минусах подхода и сравним его с монолитным стилем.**

## Что такое микросервисная архитектура?

**Микросервисная архитектура** — это способ создания программных продуктов, предполагающий разработку независимых друг от друга модулей. Каждая часть отвечает за определенную задачу и может быть изменена или расширена без перемен в других. При этом сервисы взаимодействуют между собой с помощью обмена сообщениями.

Если говорить проще, то микросервисная архитектура — это когда большая сложная система разбивается на много маленьких независимых блоков.

## Что из себя представляют микросервисы?

**Микросервисы** — архитектура на основе свободно сопряжённых сервисов с ограниченными контекстами, они нацелены на то, чтобы хорошо справляться только с одной работой.

## Как работает микросервисная архитектура?

Микросервисная архитектура состоит из отдельных, слабо связанных компонентов, каждый из которых можно разрабатывать, развертывать, эксплуатировать, изменять, не нарушая целостность приложения и работу других сервисов. Это позволяет легко и быстро развертывать отдельные возможности приложения.

В архитектуре такого типа легко экспериментировать и откатывать изменения назад, если что-то пойдет не так. Также такой формат работы ускоряет вывод новых возможностей на рынок.

## Микросервисная архитектура vs монолитная

Противоположность микросервисам — монолитная архитектура ИТ-решения, которая объединяет различные компоненты системы на одной платформе. Все части приложения в этом случае унифицированы, управление функциями осуществляется в одном месте.

**Монолит** — единый логический исполняемый файл. Он состоит из трех основных блоков:

### Пользовательский интерфейс

Он на стороне клиента. Выполняется в браузере на компьютере пользователя и состоит из HTML-страниц и javascript.

### База данных

Много таблиц, вставленных в общую и обычно реляционную базу данных.

## Серверное приложение

Приложение на стороне сервера, которое обрабатывает HTTP-запросы, извлекает и обновляет данные из базы, а также выбирает и заполняет представления HTML для отправки в браузер.

Со временем разработчики стали разочаровываться в классическом варианте архитектуры приложений.

## Среди главных минусов монолитной архитектуры:

### Сложно вносить изменения

Все части тесно связаны друг с другом — изменение, внесенное в небольшую часть приложения, требует пересборки и развертывания всего монолита.

### Единая кодовая база

Разработка ограничена изначально выбранным набором языков программирования, что затрудняет процесс вхождения в проект для новичка, которому нужно полностью изучить код системы и её функциональность.

### Невозможно масштабировать отдельный модуль

Придётся переделывать всё приложение.

**Размер базы**

Со временем кодовая база становится громоздкой

**Отказоустойчивость**

Все элементы монолитного приложения связаны друг с другом напрямую или косвенно — сбой внутри одного модуля может вызвать полный отказ системы.

В качестве альтернативы появилась архитектура микросервисов как распределенная система простых и легко заменяемых модулей. Они выполняют одну функцию и передают задачу дальше.

## Плюсы и минусы микросервисов

Микросервисы — это не таблетка от всех болезней, у подхода есть свои плюсы и минусы.

### Плюсы:

**Масштабируемость**

Благодаря независимости каждого микросервиса приложение легко масштабируется.

**Отказоустойчивость**

Проблемы внутри одного сервиса не нарушают работу системы в целом и не приводят к появлению новых ошибок.

### Гибкость стека

Для каждого сервиса можно использовать свой язык программирования, способ хранения данных, необходимые библиотеки.

### Команда может выбрать удобный язык программирования

Новый сотрудник осваивает функциональные особенности только того микросервиса, с которым ему предстоит работать — не нужно изучать систему полностью.

### Скорость релизов

Чтобы запустить новые функции или обновить существующие, достаточно изменить один модуль приложения. Это позволяет ускорить разработку и чаще выпускать обновления.

## Но есть и минусы:

### Сложно тестировать

Любые изменения интерфейса необходимо согласовывать между участниками, добавлять уровни обратной совместимости и усложнять тестирование.

**Важно постоянно продумывать взаимодействие элементов**

**Усложняется управление командами разработки и развёртывания**

Решение — применение методологий Agile и DevOps.

## Кому подойдет использование микросервисной архитектуры?

Если у вашего проекта:

- Высокий трафик и всплески нагрузки.
- Объемный код, который тяжело поддерживать и развивать.
- Множество взаимодействующих модулей.
- Различные требования к ресурсам в рамках одного приложения.
- Большая команда: работает больше 10 человек, с помощью микросервисов легче погружать новичков в работу.
- Необходимо часто делать релизы.

## Переход с монолитной на микросервисную архитектуру

### 1) Определите задачи

Каждый микросервис ограничен узкой задачей, которая решает конкретную потребность пользователя. Сформулируйте эти задачи, прежде чем начать

переход.

## 2) Соберите команду

Найдите специалистов по созданию микросервисов. Если разработчики раньше работали только с монолитами — заложите время на их переобучение. Среди плюсов микросервисов — гибкость стека. Доверьте команде выбор языков программирования и подходящих инструментов, разработчики должны отвечать за весь жизненный путь продукта. Большой плюс подхода в том, что каждая команда может создавать и развертывать свой сервис не мешая друг другу.

## 3) Создайте инструмент для взаимодействия микросервисов

В проекте нужна среда для коммуникации, поэтому необходимо организовать клиент, который поддерживает все языки программирования. Дополнительно понадобится локализованная под каждый микросервис конфигурация — с ее помощью микросервисы будут «понимать», как общаться друг с другом.

## 4) Разработка

Разработка приложения, включая время на тестирование, занимает 2-3 недели.

## 5) Ускорение разработки с помощью методологии DevOps

DevOps как нельзя лучше вписывается парадигму микросервисов. Их общие принципы – налаженная коммуникация между смежными командами, быстрый выпуск релизов.

## Какие инструменты использовать для создания микросервисов и работы с ними?

Разработка микросервисов отличается от традиционной монолитной системы.

## Наиболее популярные решения для разработки микросервисов:

**Управление API и тестирование**

API Fortress, Postman, Tyk

**Обмен сообщениями**

RabbitMQ, Apache Kafka

**Мониторинг**

Logstash

**Оркестрация**

Nomad, Apache Mesos, Kubernetes, Docker Swarm

**Разработка**

GitLab CI, TeamCity, Jenkins, Github Actions, Circle CI, Docker

**Инструменты для командной работы**

Trello, Slack, Google Meet, отечественное решение «Авандок Трекер»

## Как сделать микросервисы быстрыми и эффективными?

Ускорить работу микросервисов можно несколькими способами: при помощи кэша, распределив нагрузку в несколько потоков, с помощью асинхронного исполнения.

### Кэш

Микросервисы обрабатывают данные из разных источников. Среди них — хранилища данных, унаследованные системы и другие совместно используемые сервисы, развёрнутые локально в дата-центрах или в облаке. Все это добавляет свою задержку к времени отклика микросервиса. Кеш снижает задержку и ускоряет взаимодействие между сервисами. Архитектура микросервисов позволяет построить несколько уровней кеширования в зависимости от сценария.

### Многопоточность

Процесс из нескольких потоков, которые выполняются «параллельно». Данный подход легко внедряется, но есть и минусы — линейный рост потребления ресурсов, который можно снизить, если использовать общий для всех потоков кэш.

### Асинхронное программирование

Результат выполнения функции доступен спустя некоторое время в виде асинхронного вызова. Запуск длительных операций происходит без ожидания их завершения и не блокирует дальнейшее выполнение программы.

## Микросервисы в заказной и продуктовой разработке

Выбор архитектуры приложения зависит от целей продукта: какая ожидается посещаемость, с какими учетными системами сервис будет интегрирован, планируется ли масштабировать.

Выше мы разобрали плюсы и минусы каждого подхода для бизнеса. Дальше рассмотрим коротко. Монолит подойдет, если вы делаете стартап и у вас не предполагается роста нагрузки. Если монолитное приложение рассчитано на среднюю посещаемость в 1000 пользователей, то с ним возникнут проблемы, когда бизнес начнет расти.

Если ваша цель создание сложного высоконагруженного приложения с перспективой масштабирования, подойдет микросервисная архитектура. В долгосрочной перспективе это будет выгоднее для компании. При микросервисной архитектуре проблема масштабирования легко решается добавлением новых модулей и серверов.

## Примеры использования микросервисов

На зарубежном рынке микросервисную архитектуру используют такие гиганты, как Amazon, Twitter, Netflix, Facebook, Spotify, Uber, Google. Например, компания Netflix использует около 700 микросервисов для каждого из множества элементов, из которых состоит весь сервис. Один микросервис снимает оплату, другой хранит информацию обо всех сериалах, которые вы посмотрели, третий определяет подходящий контент.

Микросервисный подход внедряют и на российском рынке. Распространенный пример подхода — это популярные онлайн-маркетплейсы, например Avito. Каталог товаров, рейтинг пользователей, чат, отзывы — это все отдельные микросервисы в рамках одного продукта.

Эксперты компании «М.Видео-Эльдорадо» рассказывали на конференции, как применили подход в сервисе расчета цены. Раньше функционал ценообразования был разрозненным: одни задачи решал бэк-офис, за скидки отвечал другой сервис. Задача была это централизовать — сейчас стоимость на сайте, в розничном магазине или в корзине рассчитывает один сервис.