

Паттерны реализации микросервисов

Одна база данных - для одного микросервиса

Иметь общую базу данных для микросервисов (shared database) может быть привлекательной идеей на первый взгляд, но в большинстве случаев это может привести к ряду проблем и ограничений. Вот почему:

- 1. **Сложность масштабирования:** Общая база данных создает единую точку отказа. Если один из микросервисов сталкивается с высокой нагрузкой, это может повлиять на производительность всей базы данных и, следовательно, на все другие микросервисы.
- 2. **Сильная связность:** Общая база данных создает жесткую связь между микросервисами. Изменение схемы базы данных для одного микросервиса может затронуть другие сервисы, что затрудняет их независимое развертывание и обновление.
- 3. **Сложность управления данными:** Управление данными в общей базе данных может быть сложным. Разные микросервисы могут иметь различные требования к данным (например, разные модели данных и схемы), и обеспечение соответствия всем этим требованиям может быть трудоемким.
- 4. **Безопасность и доступ:** Общая база данных может потребовать более широких прав доступа, чем это необходимо для отдельных микросервисов. Это увеличивает риск утечки данных и уменьшает безопасность системы.
- 5. **Сложность резервного копирования и восстановления:** Резервное копирование и восстановление данных становятся сложными, так как одна база данных содержит данные для нескольких микросервисов. В случае сбоя, восстановление может занять больше времени и ресурсов.

Вместо этого, рекомендуется использовать паттерн, при котором каждый микросервис имеет свою собственную базу данных (паттерн "каждый микросервис - своя база данных"). Это обеспечивает большую независимость микросервисов и позволяет им развиваться, масштабироваться и обновляться независимо друг от друга. Каждый микросервис имеет полный контроль над своими данными и базой данных, что способствует улучшению сопровождаемости и масштабируемости всей системы.

*****Запомните! Иногда всё таки приходится использовать общую базу данных, например в финансовых системах где необходимо задействовать ACID транзакцию которая затрагивает бизнес-логику двух микросервисов, и нет возможности почему-то реализовать паттерн Saga(о нём позже). Но всегда помните и о недостатках такого решения.

Каждому микросервису - реализация своего кода (нет продуктовым библиотекам, которые содержат бизнес-логику)

Принцип "Каждому микросервису — реализация своего кода" подразумевает, что каждый микросервис должен быть самодостаточным в плане бизнес-логики и не должен зависеть от общих библиотек, которые содержат бизнес-логику. Это отличается от подхода, при котором общие библиотеки с бизнес-логикой используются в нескольких микросервисах. Вот несколько причин, почему этот принцип считается хорошей практикой:

Преимущества

- 1. **Независимость разработки и развертывания:** Каждый микросервис может быть разработан, развернут и масштабирован независимо. Это упрощает управление жизненным циклом каждого микросервиса.
- 2. **Легкость внесения изменений:** Изменения в бизнес-логике одного микросервиса не влияют на другие микросервисы, что уменьшает риск сбоев и упрощает тестирование.
- 3. **Локализация ошибок:** Если в одном микросервисе происходит ошибка, она не распространяется на другие микросервисы, что улучшает устойчивость всей системы.
- 4. **Гибкость технологического стека:** Разные микросервисы могут использовать разные технологии, языки программирования и инструменты, что может быть полезно для оптимизации каждого микросервиса под его специфические задачи.

Недостатки

- 1. **Дублирование кода:** Один и тот же функционал может потребоваться в нескольких микросервисах, что может привести к дублированию кода.
- 2. **Сложность управления:** Каждый микросервис имеет свою собственную кодовую базу и бизнес-логику, что может усложнить управление и сопровождение системы в целом.
- 3. **Непостоянство интерфейсов:** Разные микросервисы могут иметь разные подходы к реализации бизнес-логики, что может создать проблемы с интеграцией и согласованностью данных.

Преимущества сильно перевешивают использование этого принципа. Он подходит для систем, которые требуют высокой степени независимости и масштабируемости.

*****Запомните! Можно иметь общие технические библиотеки, которые не содержат бизнес-логику - тут они действительно ускоряют разработку микросервисов.

Также, если у вас есть core(платформенная, центральная, техническая) команда которая отвечает за то чтобы продуктовые команды не имели технических проблем - без сомнений можно даже использовать общие заготовки кода(шаблон) для создания микросервисов.

Ведь создание микросервиса — это не просто разработка функциональности, которую этот микросервис должен предоставлять. Это весьма сложный процесс, который включает в себя множество аспектов, необходимых для обеспечения надежности, масштабируемости и удобства управления. Вот некоторые из них:

- Логирование

Логирование — это процесс записи событий, которые происходят в системе. Это может быть полезно для отладки и мониторинга. Логи могут включать в себя информацию о том, какие операции были выполнены, какие ошибки произошли и так далее.

- Мониторинг

Мониторинг в реальном времени позволяет отслеживать состояние микросервиса, его производительность, использование ресурсов и другие метрики. Это необходимо для быстрого выявления и устранения проблем.

- Аутентификация и авторизация

В большинстве случаев микросервисы должны поддерживать механизмы аутентификации и авторизации(через общий сервис авторизации) для обеспечения безопасности.

- Версионирование API

Чтобы обеспечить совместимость с различными клиентами и другими микросервисами, необходимо управлять версиями API.

- Конфигурация

Микросервисы часто имеют различные настройки для разных сред (разработка, тестирование, продакшн), и управление этими конфигурациями является важной задачей.

- Сетевые политики и безопасность

Необходимо обеспечить безопасность на уровне сети, возможно, с использованием шифрования, балансировки нагрузки, ограничения доступа и так далее.