

Какие бывают кейсы создания микросервисов и как решаются?

1. Создание микросервиса с нуля, для новой функции.

- 1. Создаётся микросервис по принципам, которые обсуждали ранее
- 2. Производится интеграция между монолитом и этим микросервисом
- 3. Если нужно - производится интеграция между монолитом и фронтендом

2. Выделение фронтенда UI

- 1. Создаётся отдельное приложение UI
- 2. Производится интеграция между монолитом и этим приложением

3. Выделение микросервиса из монолита

- 1. Проанализировали границы нового микросервиса на основе принципов (которые ранее обсуждали)
- 2. Проанализировали все взаимосвязи будущего микросервиса с монолитом (то есть этой старой части монолита с другими частями монолита)
- 3. Разработали новый микросервис (он на этом этапе дублирует старую часть монолита)
- 4. Производится интеграция между монолитом и этим микросервисом
- 5. При необходимости производится интеграция микросервиса и фронтенда
- 6. Провели аналитику на данном этапе - насколько хорошо живёт ваша система в текущих условиях (пока что микросервис не включён в бизнес-процессы но работает, монолит же как обычно полноценно работает с фронтендом)
 - 1. Если всё плохо - отключаем микросервис и дорабатываем его
- 7. Если всё хорошо, двигаемся дальше. Переключаем часть трафика (клиентов) на микросервис.
- 8. Проводим аналитику - насколько хорошо всё работает в таких условиях (часть клиентов получают свои функции за счёт микросервиса, а остальная часть - за счёт монолита).

- 1. Если ошибки - то переключаем срочно всё обратно на монолит и дорабатываем микросервис

9. Если всё успешно - переключаем весь трафик клиентов на микросервис
10. Затем ждём несколько релизов монолита (чтобы точно убедиться что зависимостей не осталось со старой частью монолита, которую заменили микросервисом), и удаляем старую часть монолита
11. Готово!

*Сценарий выделения микросервиса может меняться в зависимости от ваших потребностей на проекте - например, если проект очень старый (legasy), то может делаться вокруг монолита "обёртка" из кода, который будет взаимодействовать с новыми микросервисами.

Но вы спросите - а как можно переключать часть трафика безболезненно? Есть такой инструмент - фича-тоггл.

("feature toggle") — это технический приём, позволяющий динамически включать или отключать определенные функции программного обеспечения без необходимости изменения кода и перезапуска приложения.

Простой пример:

Представим, у вас есть интернет-магазин, и вы хотите добавить новую функцию "Быстрый просмотр товара". Эта функция позволит пользователям просматривать подробности товара прямо на странице каталога, без перехода на отдельную страницу.

Шаги с использованием фича-тоггла:

1. **Разработка:** Вы разрабатываете функцию "Быстрый просмотр", но вместо того чтобы сразу её активировать, вы заворачиваете её в фича-тоггл. В коде это может выглядеть примерно так:

```
if (featureToggle.quickViewEnabled) //Буквально-если переключатель включён,то выполни код в блоке ниже
{
    // Код для функции "Быстрого просмотра"
}
```

2. **Деплой:** Вы деплоите обновление на сервер, но функция "Быстрый просмотр" ещё не активна для всех, потому что `featureToggle.quickViewEnabled` установлено в `false`.

3. **Тестирование:** Вы включаете фича-тоггл (`featureToggle.quickViewEnabled = true`) для небольшой группы пользователей или на тестовом сервере. Это позволяет вам убедиться, что новая функция работает, как нужно.

4. **Активация для всех:** Если тестирование прошло успешно, вы включаете фича-тоггл для всех пользователей. Если выявлены ошибки — исправляете их и снова проводите тестирование.
5. **Удаление фича-тоггла:** После того, как вы убедились, что функция работает стабильно, фича-тоггл можно удалить из кода, сделав функцию постоянно доступной.

Таким образом, фича-тогглы дают вам гибкость и контроль над внедрением новых функций, минимизируя риски. И также можно переключать трафик на микросервисы, создавая фича-тогглы типа `"featureToggle.microserviceOrderEnabled"`

Какие бывают способы миграции базы данных из монолита в микросервисы?

1. DBlink SQL скрипт

Как это работает:

С использованием скрипта на SQL с DbLink или другими средствами для удаленного соединения с базой данных, вы создаете запросы, которые могут перенести данные из одной базы в другую.

Плюсы:

- Простое и быстрое решение для малых и средних объемов данных.
- Может быть автоматизировано.
- Не требует изменений в приложении.

Минусы:

- Могут быть ограничения со стороны баз данных, например, с точки зрения безопасности.
- Если данные очень большие, это может быть неэффективно и затратно по времени.

2. Batch-скрипты

Как это работает:

На уровне приложения создается скрипт (batch job), который извлекает данные из старой базы и добавляет их в новую базу данных микросервиса.

Плюсы:

- Больше контроля над процессом миграции.
- Можно включить бизнес-логику и валидацию данных прямо в процессе миграции.

Минусы:

- Если данных слишком много, это может быть неэффективно.
- Потребуется дополнительное время и ресурсы на написание, тестирование и поддержку скрипта.

3. Read-is-a-Move (Чтение как перемещение)

Как это работает:

Эта стратегия предполагает двойное чтение: сначала из новой базы, а если данных там нет — из старой. При этом, если данные найдены в старой базе, они переносятся в новую.

Плюсы:

- Позволяет очень эффективно переносить огромные объемы данных, так как перенос осуществляется "на лету" при обращении к данным.
- Минимизирует риск "простоя" или недоступности данных.

Минусы:

- Сложно реализовать и поддерживать, так как требует двойного чтения и логики переноса данных.
- Возможны проблемы с согласованностью данных между двумя базами.

В зависимости от ваших потребностей, один из этих методов может быть более подходящим для вашей задачи.