

Сначала о цели микросервисной архитектуры и о связанности/сплочённости

Цель внедрения микросервисов - ускорить процесс вывода готового продукта (нового кода) до клиентов. Это достигается за счёт реализации принципа низкой связанности.

Принципы "Низкая связанность (Low Coupling)" и "Высокая сплоченность (High Cohesion)" являются ключевыми для разработки программного обеспечения, независимо от того, используете ли вы монолитную архитектуру или микросервисы. Однако эти принципы могут проявляться по-разному в зависимости от выбранного подхода. И естественно гораздо ярче проявляются при использовании микросервисной архитектуры.

Низкая связанность (Low Coupling)

Этот принцип подразумевает минимизацию зависимостей между различными частями системы. Например - сервис А создаёт заказ, а сервис Б проводит его оплату, и никак иначе. Сервис А не знает ничего о процессе оплаты, а сервис Б не знает и не должен знать ничего о процессе создания заказа.

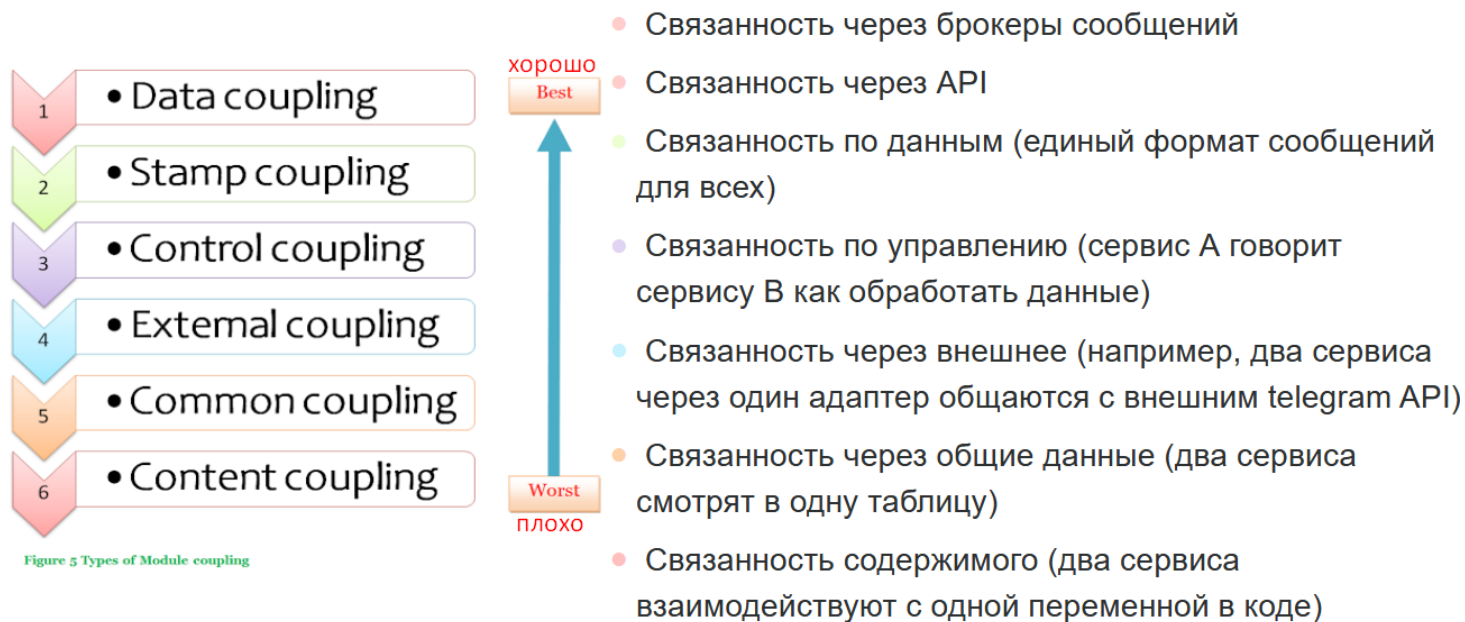
В контексте монолита:

- Стремление к низкой связанности может привести к более модульному монолиту, где различные функциональные области слабо связаны между собой.
- Это может быть реализовано через разделение кода на слои и т.д.

В контексте микросервисов:

- Низкая связанность означает, что каждый микросервис должен иметь минимум зависимостей от других микросервисов.
- Это упрощает разработку, тестирование и развертывание каждого микросервиса отдельно.

Посмотрите общие критерии связанности для микросервисов:



Высокая сплоченность (High Cohesion)

Принцип высокой сплоченности говорит о том, что код, относящийся к одной функциональной области, должен быть организован в одном месте. То есть, если вы реализовали сервис А для создания заказа - то код сервиса А должен состоять только из того, что необходимо для создания заказа и ничего более.

В контексте монолита:

- Это может быть реализовано через разделение кода на модули или пакеты, каждый из которых решает конкретную задачу.
- Высокая сплоченность упрощает понимание и изменение кода, поскольку все, что относится к одной функциональной области, находится в одном месте.

В контексте микросервисов:

- Каждый микросервис должен быть высоко сплоченным, то есть отвечать за одну и только одну функциональную область.
- Это делает микросервисы гибкими и независимыми, позволяя разработчикам быстро вносить изменения в отдельные части системы.

Посмотрите общие критерии сплочённости для микросервисов:

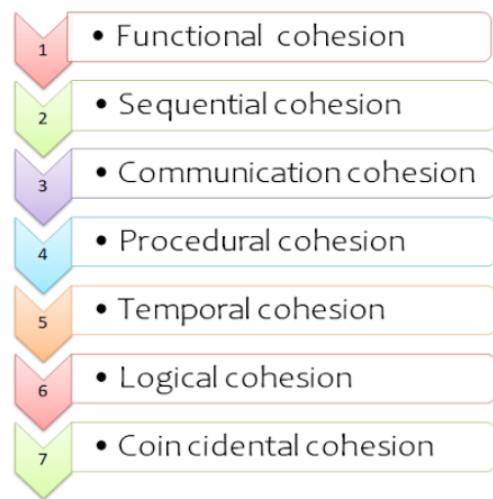


Figure 7 Types of Module cohesion

хорошо
Best



Worst

плохо

- Функциональная (части модуля разделены на решение конкретной задачи)
- По последовательности действий (результат работы одной части модуля это входные данные для другого метода (методы по цепочке))
- По взаимодействию (части модуля работают над общими данными)
- Процедурная (части модуля можно использовать только в определённой последовательности)
- Временная (части модуля вызываются в одно и тоже время)
- Логическая (части относятся к одной проблеме, но разные по природе)
- Случайная (сгруппированы случайно)

Итоги применения принципов:

1. **Масштабирование:** Микросервисы, следующие принципам низкой связанности и высокой сплоченности, легче масштабируются, поскольку каждый из них можно развертывать независимо.
2. **Разработка и Развертывание:** В монолитах, даже если они хорошо спроектированы с учетом этих принципов, изменение одного модуля часто требует пересборки и перезапуска всего приложения. В микросервисах такой проблемы нет.
3. **Комплексность:** Низкая связанность и высокая сплоченность могут сделать монолит более управляемым, но по мере роста системы увеличивается сложность. В микросервисах каждый сервис остается относительно простым, поскольку отвечает только за одну функциональную область.
4. **Технологический стек:** Микросервисы позволяют использовать для каждого сервиса наиболее подходящие технологии, что может быть сложно или невозможно в монолитах.

В итоге, принципы низкой связанности и высокой сплоченности в случае микросервисов проще для реализации и могут принести больше преимуществ, и помочь достигнуть цели внедрения микросервисной архитектуры - быстрее производить готовый продукт