Существуют два подхода к архитектуре приложения – **монолитный** и **микросервисный**.

 

**В монолитной архитектуре** приложение можно разделить по слоям, но его части все равно тесно связаны между собой (их пишут внутри одного проекта и собирают одновременно). Подходит для небольших приложений, которые не будут развиваться и усложняться.

Преимущества – скорость разработки, линейный процесс деплоя, полный взаимный доступ частей приложения, легкая поддерживать транзакционность процессов, единая БД.

Недостатки – сложности изменения или добавления нового функционала и технологий, проблемы с масштабированием, много кода в одном проекте, сложно разобраться в проекте.

**В микросервисной архитектуре** приложение делится на сервисы, которые отвечают за отдельную функциональность и общаются между собой. Каждый микросервис связан со своей БД, которая хранит только те данные, которые относятся к его функциональности. К другим БД у микросервиса доступа нет.

Преимущества – легкость изменения или добавления нового функционала и технологий, масштабируемость, легче переиспользовать код, удобнее распределять задачи при разработке.

Недостатки – скорость разработки, задержки и ошибки при общении микросервисов между собой, ниже производительность, сложно поддерживать транзакционность процессов, несколько БД, сложный процесс *CI/CD* (деплоя).

Межсервисные взаимодействия могут быть синхронными и асинхронными.



При **синхронном взаимодействии** задачи выполняются друг за другом. На примере - сверху вниз.

Недостатки:

* Клиенту нужно дождаться ответа на запрос, чтобы убедиться, что все прошло успешно. Иногда это занимает много времени, при этом процесс загружен ожиданием, что отнимает ресурсы.
* Приходиться продумывать сценарии обработки долгих запросов. Чтобы не ждать вечно, время ожидания ограничивают.
* Если клиент выполняет слишком много запросов, сервер может не выдержать нагрузки.



При **асинхронном взаимодействии** между клиентом и сервером используется посредник – **брокер** **сообщений** (например, *Apache Kafka* или *RabbitMQ*). Он принимает запросы от сервисов в сформированные ею очереди, сохраняет у себя, а затем передает другим сервисам.

Преимущества:

* Процессы не блокируются, потому что клиент не ждет ответа от сервера.
* Сервер не перегружен, а все избыточные запросы остаются в очереди.
* На очередь могут подписаться несколько серверов. Например, и микросервис по отправке писем, и микросервис по возврату средств.
* Не нужно продумывать сценарии обработки долгих запросов. Клиент не ждет ответа.

Недостатки:

* Нужно разворачивать и настраивать дополнительный сервис (брокер сообщений).
* Если запросы попадают в очередь быстрее, чем их успевает обрабатывать сервер, то рано или поздно закончится допустимый объем хранимых сообщений.