

- **Async Request-Reply**

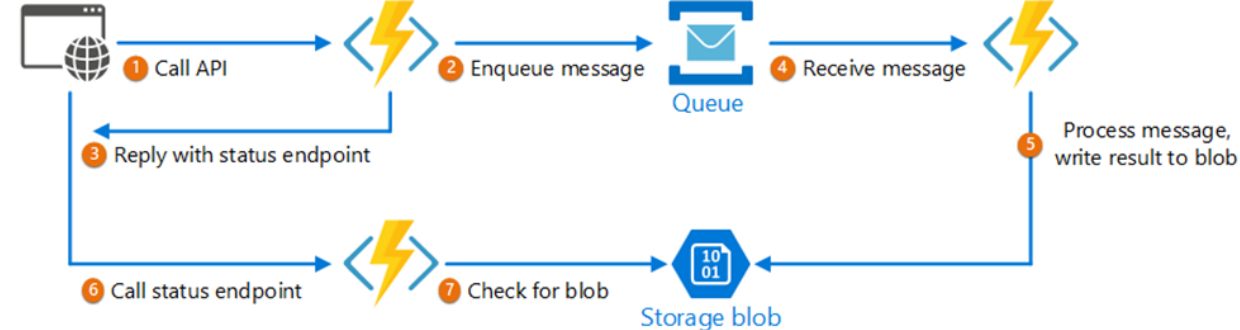
Async Request-Reply — это паттерн, который используется для асинхронного взаимодействия между клиентом и сервером через API (при этом первый запрос - синхронный). Вместо того, чтобы ждать ответа сервера после отправки запроса, клиент получает уведомление о том, что запрос принят, и может продолжить работать, пока сервер обрабатывает запрос. Ответ будет отправлен клиенту позже, когда сервер завершит обработку. Этот подход улучшает производительность и масштабируемость системы, особенно при обработке долгих запросов или при высокой нагрузке.

Основные характеристики:

1. Асинхронное взаимодействие между клиентом и сервером, после первого синхронного запроса.
2. Отсутствие блокирования клиента во время обработки запроса на сервере.
3. Обычно используется очередь сообщений для асинхронной обработки запросов и ответов, но могут быть вариации.
4. Позволяет клиенту обрабатывать другие задачи, пока сервер выполняет запрос.

Недостатки:

1. Сложность: асинхронное взаимодействие требует больше усилий для реализации и поддержки, чем синхронный подход.
2. Отслеживание состояния: клиенту и серверу может потребоваться отслеживать состояние запросов и ответов, что может увеличить сложность системы.
3. Требования к инфраструктуре: необходимость использования дополнительных компонентов, таких как очередь сообщений, что может увеличить затраты на инфраструктуру и сложность поддержки.
4. Задержка в обработке ответов: хотя асинхронность позволяет клиенту продолжать работать без ожидания ответа, это может привести к задержкам в получении ответов из-за очередей или загруженности сервера.



- Как видите, сначала идёт 1 синхронный запрос REST, на который сразу приходит ответ 3 - 201 Created (например, мы приняли задание на создание заявки).
- Потом через очередь сообщений (2 и 4) (или могут быть другие асинхронные технологии) задание передаётся в сервис, который должен создать заявку.
- После, 5 шагом результаты сохраняются в некое хранилище (Blob - это сервис для хранения больших объемов неструктурированных данных, доступ к которым можно получить по протоколу HTTP).
- А дальше уже сам клиент 6 шагом опрашивает Blob на наличие результата (создана заявка или нет).
- Если создана — тогда асинхронно у пользователя в интерфейсе появляется результат создания заявки.
- Если нет — опрос продолжается.

Таким образом, совмещая первый синхронный запрос и следующие асинхронные процессы, мы получаем гибкую систему, где пользователь не должен ждать результата создания заявки прямо сейчас, а в фоне может делать свои дела (при этом мы уверены что точно подали заявку через синхронный вызов).

Когда возможно нужно применять этот паттерн:

1. Когда обработка запросов занимает значительное время, и блокирование клиента на время ожидания ответа нежелательно.
2. Когда у вас есть высоконагруженная система, и вы хотите оптимизировать производительность и масштабируемость.
3. Когда система должна обрабатывать различные виды запросов с разными временем обработки, и вам нужно гарантировать, что быстрые запросы не будут заблокированы медленными.

4. Когда вы хотите предоставить возможность клиентам отменять долгие запросы, не дожидаясь их завершения.

Когда возможно не нужно применять этот паттерн:

1. Когда обработка запросов происходит быстро, и синхронное взаимодействие между клиентом и сервером подходит для задачи.
2. Когда важнее простота реализации и поддержки, а не максимальная производительность и масштабируемость.
3. Когда нет необходимости в отслеживании состояния запросов и ответов, и клиент должен получать ответ немедленно после отправки запроса.

Реальный пример:

Amazon Web Services (AWS) использует асинхронные API для некоторых своих сервисов, таких как AWS Lambda и Amazon Simple Notification Service (SNS). Эти сервисы обрабатывают запросы асинхронно, позволяя клиентам продолжать работать без ожидания ответа от сервера. Асинхронность позволяет AWS оптимизировать производительность и масштабируемость своих сервисов и улучшить взаимодействие с клиентами.