Digital Solution
Architecture

Synchronous and Asynchronous and everything in between

Decision Drivers

Quality Attributes That Shape API Design

Scalability & Throughput

System capacity to handle increasing load and concurrent operations without degradation

Latency & Notification Speed

Time delay between operation completion and client awareness of results

Reliability & Fault Tolerance

Guaranteed delivery and graceful handling of failures across distributed components

Security & Trust Boundaries

Authentication, authorisation, and protection across organisational boundaries

Maintainability & Observability

Ease of debugging, monitoring, and evolving the system over time

Client Simplicity

Implementation complexity and compatibility requirements for consuming applications

Synchronous or Asynchronous?

Synchronous

- Easy programming model
- Easier error handling
- More coupling
- Strict data consistency achievable

Asynchronous

- More levels of indirection allow for decoupling
  - Decoupling of uptime
- More robust in certain situations
- More difficult programming model
- Eventual consistency as only option

ASYNC DESIGN CHOICES

Option A: Polling

How It Works

Server immediately returns `HTTP 202 Accepted` with a status URL. Client periodically queries this endpoint to check operation progress until completion.

Advantages

- Simple client implementation
- Firewall and proxy friendly
- Strong security boundaries
- No exposed client endpoints required

Trade-offs

- Additional server load from repeated requests
- Delayed result awareness
- "Thundering herd" risk at known intervals
- Inefficient for time-sensitive operations

Best suited for: Clients unable to expose public endpoints, such as mobile applications or services behind restrictive firewalls.

Option B: Callback / Webhook

How It Works

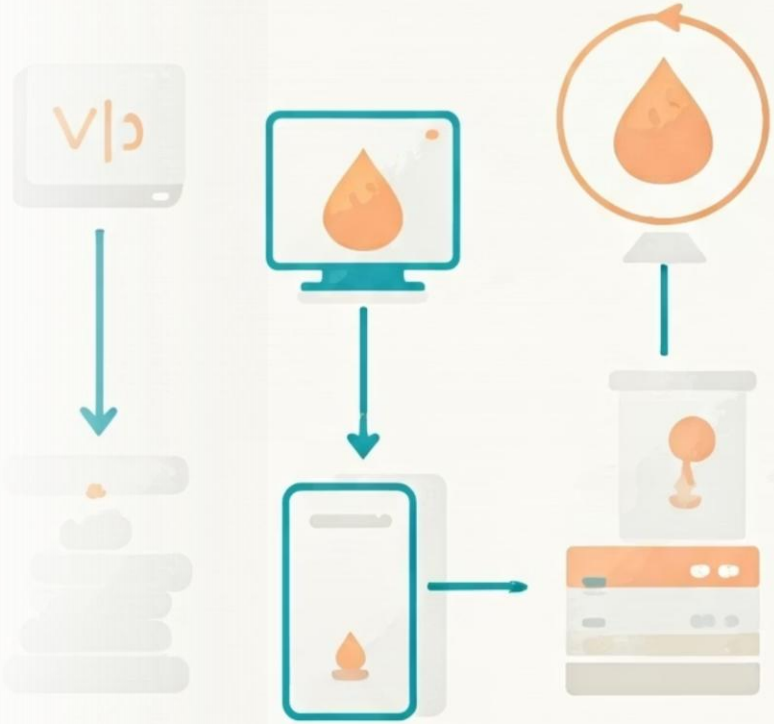
Client provides a callback URL during request initiation. Server actively invokes this endpoint when the operation completes, pushing results directly to the client.

Advantages

- Minimal latency between completion and notification
- Server-efficient: no status endpoint needed
- Ideal for server-to-server integration
- Enables immediate downstream processing

Trade-offs

- Client must expose and secure public endpoint
- More complex authentication mechanisms
- Retry and delivery guarantee logic required
- Firewall and network topology constraints



Best suited for: Advanced clients with robust infrastructure, particularly internal services or trusted integration partners.

Option C: Push / Event Stream

How It Works

Server publishes results to an event broker (Kafka, RabbitMQ) or maintains persistent connections (WebSocket, Server-Sent Events). Multiple subscribers receive updates in real-time.

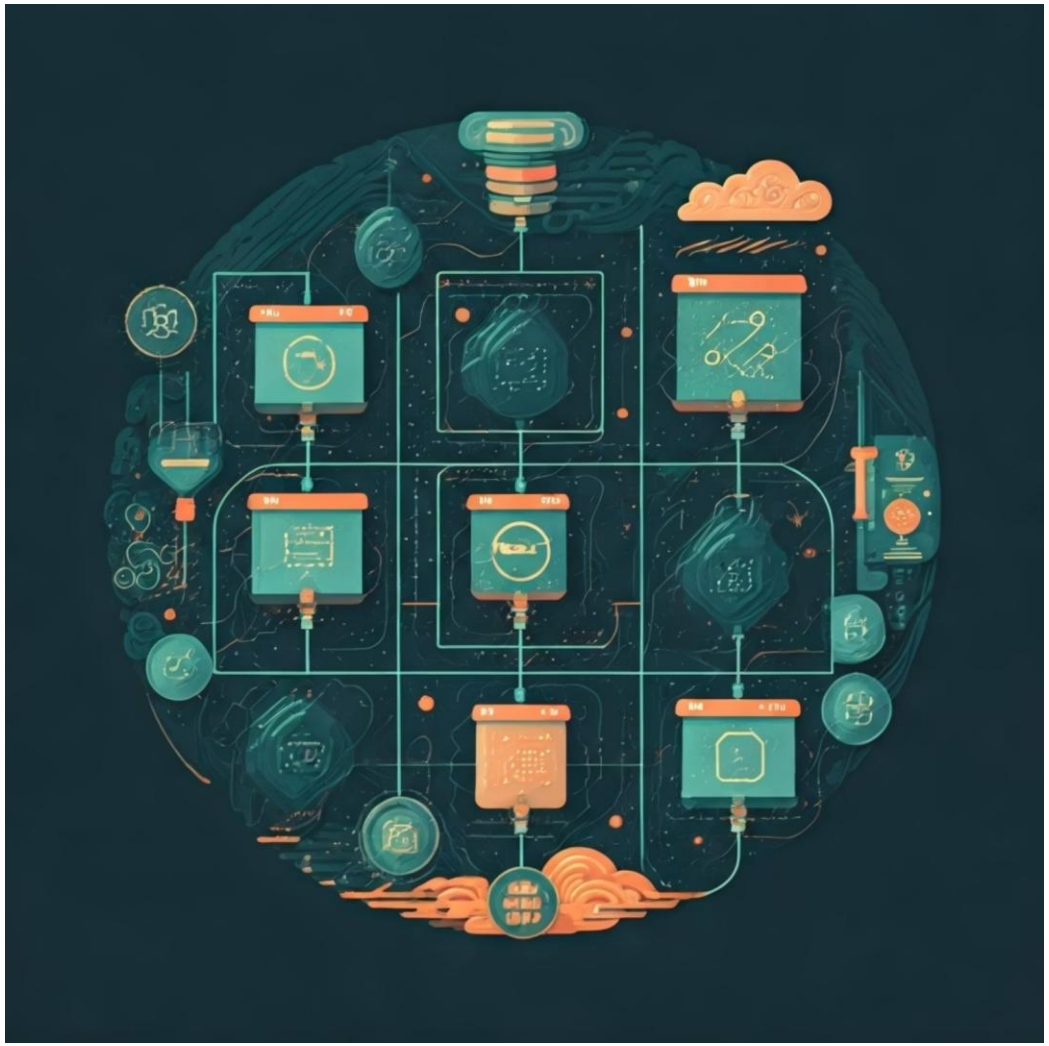
Advantages

- True real-time notification delivery
- Highly scalable for many subscribers
- Natural fit for event-driven architectures
- Decouples producer and consumer lifecycles

Trade-offs

- Complex infrastructure requirements
- Persistent connection management overhead
- Event ordering and delivery semantics
- Operational expertise needed for brokers

"Event streams excel when multiple downstream systems need immediate awareness of state changes."
Best suited for: Streaming scenarios, multi-client broadcasting, and systems already invested in event-driven infrastructure.



What do you use?

What challenges do you face?

What decision drivers do you see?