Service Interactions

Options to establish communication between ServiceA and Service B:

- Request-Response communication with HTTP (often REST)
- Message-based / Event-driven communication

Request-Response Communication With Http	Message-Based / Event-Driven Communication
Pros	
HTTP itself is a request/response protocol and synchronous in nature, so REST is a great fit for request-reply interactions. HTTP and REST approaches are most common, especially for public services) and are - inter-operable with every programming language - easy to document (with tools like Swagger and adherence to standards)	*Here, communication is asynchronous in nature and delivers a truly decoupled architecture where services are independent of each other, resulting in and ensuring* - Non-blocking of services - Data (eventual) consistency / integrity across services - Ease in scaling
Cons	
Regardless of code execution at the client side (sync/async), the client can only continue task when it receives response from server side. *As a result, the services are still coupled in a sense and are just simply choreographed using REST communication. Some cons:* - Blocking / increased latency (proportionate to HTTP call chains) - Increased likelihood for data inconsistency (e.g in events of service failure during HTTP call) - More testing/debugging (in situations of change to a Service)	*Services do not communicate directly with each other, but rely on an abstracted component - the message broker/service bus, for communication.* *This creates the need to manage this component attracting extra complexity, costs and effort. Some challenges* - Simultaneous state management of original service (publisher) while resiliently publishing its related integration event - Ensuring message idempotence and deduplication implementations in broker and/or services

- Scaling problems	- Ensuring availability, reliability and data integrity (including disaster recovery) with service scale
Best Use Case	
Request-response scenarios with short response time and (usually) a single receiver. E.g querying data for real-time/live user interfaces (front-end service)	When any/all conditions - response time is long/unknown - business tasks span across multiple services - (multiple) services need to be aware of certain event(s) - eventual consistency required across services