Services Interaction

Imagine that we have two services - ServiceA and ServiceB. Both services have to communicate with each other in some way.

Given this information, please answer following questions:

1. What options do we have to establish such communication?

- HTTP Communication
    - Communication via HTTP over the internet via Rest APIs. IP addresses of the 2 servers where the services are hosted become the unique identifier or addresses to make a connection between them using any protocol (TCP/IP, SMTP, HTTP, HTTPS, UDP, FTP etc.)
        - Here in this case both services are heavily dependent on each other, they are tightly coupled to each other and have to wait for the response in order to perform any action. This flow is also called Synchronous communication as both services have to wait for the response to proceed further.
        - We can also consider HTTP asynchronous communication where one service doesn't wait for a response instead they are responded with some URL or a Promise and then the respective service checks the progress and resolve or wait until the response is completed and the communication is established.

- Message-based Communication
    - In this communication services do not communicate directly to each other, the client service sends requests but doesn't wait for a response. It is a completely loosely coupled communication where there is no need of direct connection between 2 services
        - Here client service sends a message to broker systems (RabbitMQ, SQS etc.) and the message producer doesn't wait for a response and on consumer side service listens for message events asynchronously. It is also called as a Publisher/Subscriber Design pattern where one service

will act as publisher and the other will act as a subscriber. This eliminates a lot of complexity associated with HTTP communication.

- Event-driven communication
    - This is another asynchronous communication where both services heavily rely on communication via events, unlike message communication where both service must know the message payload structure to take actions. Message broker is required though for the data transfer
        - Services (Service A and B) respond to the events occurred by listening to the events produced by the other service
        - In Event driven communication, both services doesn't need to know any message structure and have to publish events with a payload and consuming services don't need to know any details for the message, they just need to take actions on those events. Both services will only respond to events and don't have to take care for message delivery.

2. For each option describe what are the pros and cons of this solution?

- HTTP Communication
    - Pros
        - Synchronous Request/Reply
        - Easy to understand and maintain
        - Set of Protocols to abide across the web to use it across different applications
        - Simplified and common schema and options to communicate with any number of services
    - Cons
        - Tightly coupled - Service A have to fully depend on Service B to take any actions on requests made.
        - Blocking - Here services are blocked waiting for response, this can affect performance as same thread ca be used for other processing instead of just waiting for response.
        - Retry/Error handling - If Service B fails or if it's not able to respond then Service A needs to have some sort of retry mechanism.
- Message Communication

    - Pros
        - Loosely coupled
        - Non-Blocking - Here services don't have to wait for a response
        - Fault tolerance - If consumers are down then the application will continue to work and messages are saved in the queue to be picked up later on.
    - Cons
        - Some coupling is present as both services must agree on the message payload structure
        - Error handling is difficult
        - Less fault tolerance due to loss of message or queues
- Event driven communication
    - Pros
        - Loose coupling
        - Non-Blocking
        - Fault tolerance
        - Cost-effective - No need to do continuous polling
    - Cons
        - Error handling
        - Bit of complex setup (Setup queue/event system, prepare services to listen for events and take action)

3. For each option describe what are the cases the solution fits best?

- HTTP Communication
    - Rest APIs - Most of the applications (Web apps, Mobile Apps) uses the Rest APIs for Client to Server communication
    - Static & Dynamic Websites - Websites built where the data is transferred from one server to the browser is fulfilled by the browser only via HTTP or HTTPS

- Message Communication
    - Sending bulk emails - Messages are queued and can be sent asysnchronously via different service

    - Live Chat - Chat apps using in-app or web interfaces to let users send messages to single or multiple subscribers is based on Pub/Sub pattern
    - Data Analytics - Aggregate data and push in a message broker and can be used to create visualisations on aggregate data which cannot be saved in the database
    - Event Buses - Message brokers help with asynchronous event driven communication where payload is unknown and help in passing data from one place to another

- Event-driven communication
    - Notifications and Updates in any Application
        - Push Notifications
        - Emails
        - Slack, Whatsapp, SMS etc.
    - Distributed Caching - Seeding Caches parallelly without depending upon client to burst or seed the cache in distributed cache systems
    - Distributed Logging - Tracking critical events and listening them to log in distributed manner