1) Synchronous vs Asynchronous Communication

Firstly, let me explain these terms from etymological aspect. The word “synchronous” is originating from the Greek language. The prefix "Syn-" (συν-) means "together" or "with” and "Chronos" (χρόνος) means "time", so the word “synchronous” can be translated as “happening simultaneously or happening at the same time”. From that perspective, we can easily conclude that the word “asynchronous” means not happening at the same time due to the Greek prefix A-" (ἀ-) means "not" or "without. If we put aside the etymology, In a microservices architecture, services are designed to be loosely coupled while still needing to communicate with each other to maintain application consistency. In this context, we primarily encounter two different types of communication: synchronous and asynchronous communication.

Synchronous communication waits for a response after sending a request to another service and blocks any operation with any microservice before getting a response. HTTP is by far the most well-known and widely used protocol as an example of synchronous communication.Let's assume we want to perform an action through an application or a web browser. Our applicaiton or web server sends a related http request to the related data such as GET,POST,etc. to the server. Then servers are aiming to fullfill the request with respect to the content of the related method like retrieving or saving data by interacting other APIs, databases. After processing the request, the server sends an immediate HTTP response that includes an HTTP status code indicating the success or failure of the request. The response also contains a body with relevant data, if applicable. Finally, our client (whether it's a web server or an application) takes action based on the received response. This request/response loop continues accordingly, with the client and server **waiting for each other** as needed, allowing for the interaction and flow of data between them.

On the other hand, asynchronous communication eliminates the need to wait for a response and allows processes to continue performing other tasks without blocking. Asynchronous communication is managed by message brokers such as RabbitMQ, Kafka, etc. For instance, a task queue system like Celery paired with Redis or RabbitMQ is employed to queue

background jobs, enabling efficient handling of tasks without halting other processes.

Microservices enqueue tasks in an asynchronous manner, while dedicated worker processes retrieve tasks from the queue asynchronously. These workers then execute tasks in the background, providing updates on job status and notifying other services upon completion. Enqueuing tasks without any delay enables microservices to operate independently in a loosely coupled manner. Similarly, message queues also serve as a good example of asynchronous communication in distributed systems. Event-Driven Architecture, Asynchronous APIs with Callbacks, and Batch Processing are additional examples of patterns that we might encounter in asynchronous communication.