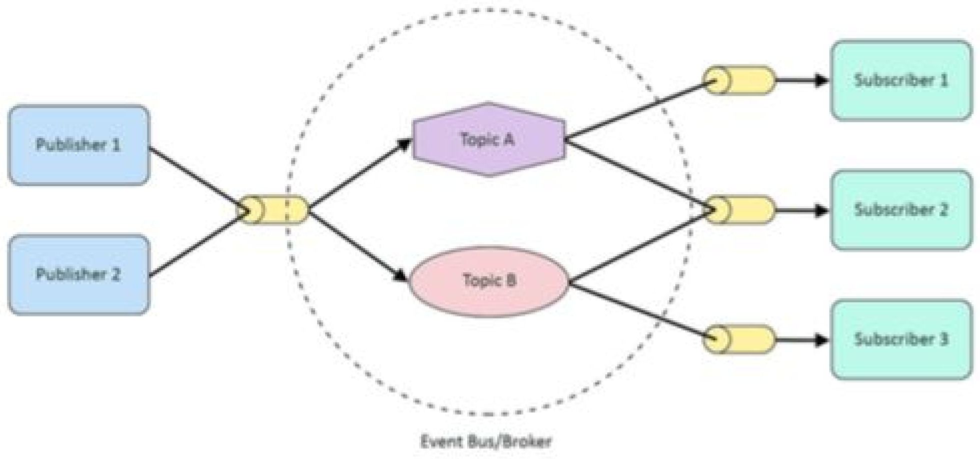
Publish-Subscribe architecture

Definition

- The Publish-Subscribe architecture is a messaging pattern that decouples the entities sending the messages and the ones listening for the messages. It allows publishers (senders) to broadcast messages to several subscribers (receivers) without them being tightly coupled. Essentially, the publishers are unaware of which application is going to receive the message whereas the subscribers do not really care about who actually sent it.
- **Vocabulary for components and connectors:***Publishers* :
    - Broadcasts messages with no knowledge of who is subscribing.
    - *Eventbus/Broker* : Transfers messages from publisher to subscriber; holds messages until they are taken by subscribers or they timeout.
    - *Subscribers* : "Listens" for messages they are interested in without knowing who is sending the messages (publishers). Acknowledges when it has taken a subscribed message from the bus.

Topological constraints



(The figure above shows a publish-subscribe model with 2 publishers, an event bus and 3 subscribers. Subscriber 1 is subscribed to Topic A so it will receive messages with Topic A. Subscriber 2 is subscribed to Topic A and Topic B, thus it will receive messages with both topics. Lastly, Subscriber 3 is subscribed to Topic B so it will receive messages with Topic B.)

- This architecture involves multiple publishers and multiple subscribers.
- Multiple subscribers can subscribe to multiple topics.
- Communication is one way , from the publishers to the subscribers via the brokers.

Applicable problems

● **Applications that require N to N scalability**: Pub-Sub architecture supports many different communication patterns. Those patterns include 1 to 1 , many to many , 1 to many , and many to 1 . This means each subscriber subscribes to at least one publisher and each publisher sends messages to at least one subscriber. These loosely coupled components allow for more publishers and subscribers to scale quickly and dynamically.

● **Need for independent operations**: Publishers do not need to know of a subscribers existence. Since publishers post to topics, and subscribers subscribe to topics, neither need to know the system topology and can operate independently of one another.

Resilience to change

- This architecture is resilient to many changes. Adding or removing topics is convenient and scalable because it can be done without changing the architecture.
- Adding and removing both publishers and subscribers are seamless.

Negative behaviours

● **Bottlenecks:** The eventbus/broker is the bottleneck of the system. This means as the system scales the performance of the broker is proportionate to the performance of the system.

● **No Guaranteed Stability**: Since publishers and subscribers do not have any knowledge of each other, publishers can not guarantee that all messages have been sent properly. The stability of the messages being sent is dependant on the broker or bus delivering the messages properly. This issue can be mitigated by forcing subscribers to send a handshake or acknowledgement of receipt to publishers.

- **Lack of Message Flexibility:** After the creation of a message data structure, modifying the message format can be complex. This is due to the fact that modifying the message structure results in all users of that message having to be altered to accept the new format. If subscribers are external, this may be impossible. Possible mitigation strategies include sending messages with a version number so that subscribers can ensure they are receiving the correct message format.

- **Asynchronous Publishing:** If a subscriber begins subscribing to a topic after its initial inception, messages posted to that topic prior to the subscription may be lost.

Supported NFPs

- Scalability:
  - Supported by the decoupled nature of the architecture, allowing additional publishers and subscribers to be added efficiently without modifications to the rest of the system. However, scalability is inhibited by the slowdowns that might occur when there are many subscribers to a topic.

- Adaptability:
  - Supported again by the decoupled nature of the architecture, allowing new publishers, subscribers and topics to be added to the system without modifications to existing components. Additionally, implementations and algorithms of individual components can be modified without affecting upstream or downstream components. However, adaptability is inhibited when the structure of existing message data needs to be modified.

- Dependability:
  - Inhibited by the fact that messages are not guaranteed to be delivered to subscribers due to the lack of message delivery acknowledgement in the architecture.

Inhibited NFPs

- Flexibility :
  - Once the structure of the payload is decided, it is hard for it to be changed in the Pub-Sub pattern
- Stability:
  - Due to the decoupled nature of the publish-subscribe architecture, two scenarios arise where message delivery to subscribers may fail. If an event bus is designed to deliver messages for a specified time and stop sending messages thereafter, there is no

guarantee that all subscribers receive the message because there is no acknowledgment mechanism in the architecture. Another scenario where message delivery fails is caused by the fact that publishers assume that subscribers are listening even when they are not. Publishers will not know that there are no subscribers listening and important messages can be lost.

Comparison with other architectures

- Event-based architecture is not subscribing to a generator, rather than a bus and Publish-subscribe architecture involves a bus that acts as an intermediary between the publisher and subscriber.
- Unlike client server architecture, two way communication is not possible in publish subscribe architecture.

Credit:

Students:

Ripe (Mena Labib, Shakaib Khan, Zuheir Al Sagha, Mark Emery)

CannaBuds (Brandom Lam, Shiyu (Alexis) Zhang)

Instructor and TAs:

Mei Nagappan, Achyudh Ram Keshav Ram, Aswin Vayiravan, Wenhan Zhu