PARADIGMA PUBLISHER/SUBSCRIBER

Il modello Publisher/Subscriber, o Pub/Sub, è un paradigma di messaggistica asincrona e disaccoppiata in cui i mittenti di messaggi (i "publisher") non inviano direttamente i messaggi a specifici destinatari (i "subscriber"), ma piuttosto a categorie di messaggi, chiamate "topic" o "canali". I destinatari che sono interessati a certi tipi di messaggi si "iscrivono" (subscribing) a questi topic, e ricevono automaticamente tutti i messaggi pubblicati su di essi. Un'applicazione client/server si può trasformare sempre in un'applicazione pub/sub dove i client e server originari diventano più leggeri e la complessità viene scaricata sull'host che fa da broker. I componenti chiave del modello Pub/Sub sono:

1. Publisher (editore):
   - è l'entità che crea e invia messaggi;
   - non ha conoscenza dei subscriber; non sa quanti ce ne sono, né chi sono.
   - si limita a "pubblicare" un messaggio su uno specifico topic o canale.
2. Subscriber (iscritto):
   - è l'entità che riceve i messaggi;
   - dichiara il proprio interesse per uno o più topic specifici (si "iscrive");
   - non ha conoscenza dei publisher; non sa chi sta inviando i messaggi;
   - riceve tutti i messaggi pubblicati sui topic a cui è iscritto.
3. Topic/Channel (argomento/canale):
   - è una categoria o un "nome" astratto al quale i publisher inviano i messaggi e i subscriber si iscrivono;
   - rappresenta il "mezzo" attraverso cui i messaggi vengono filtrati e distribuiti.
4. Message Broker/Event Bus (broker di messaggi/bus di eventi):
   - questo è il componente fondamentale che rende possibile il disaccoppiamento;
   - è un intermediario che riceve i messaggi dai publisher e li inoltra ai subscriber appropriati;
   - è responsabile della gestione dei topic, delle iscrizioni e della distribuzione efficiente dei messaggi.

Come Funziona, a livello di flusso di lavoro:

1. un Subscriber si connette al Message Broker e si iscrive a un Topic X;
2. un Publisher si connette allo stesso Message Broker e pubblica un messaggio sul Topic X;
3. il Message Broker riceve il messaggio dal Publisher;
4. il Message Broker identifica tutti i Subscriber iscritti al Topic X;
5. il Message Broker inoltra il messaggio a tutti i Subscriber interessati.

Vantaggi:

1. Disaccoppiamento (Decoupling):
   - temporale: Publisher e Subscriber non devono essere attivi contemporaneamente. Il broker può conservare i messaggi (o distribuirli appena un subscriber torna online, a seconda dell'implementazione);
   - spaziale: Publisher e Subscriber non devono conoscersi reciprocamente o sapere dove si trovano. Il broker fa da intermediario;
   - tecnologico: possono essere implementati in linguaggi e su piattaforme diverse, purché comunichino con lo stesso broker. Questo disaccoppiamento rende il sistema più robusto, flessibile e facile da scalare e mantenere.
2. Scalabilità:
   - è facile aggiungere nuovi Publisher o Subscriber senza modificare quelli esistenti;
   - se un topic ha molti subscriber, il broker gestisce la distribuzione a tutti loro;
   - se un publisher invia molti messaggi, il broker può gestirli e metterli in coda;
3. Affidabilità: il broker può garantire la consegna dei messaggi (a seconda della configurazione, es. "almeno una volta", "esattamente una volta");
4. Flessibilità: permette facilmente di implementare notifiche one-to-many (un publisher, molti subscriber) o molti-a-molti (molti publisher, molti subscriber). Di fatto, va ricordata come una delle differenze più prepotenti rispetto al modello client-server, che invece lavora necessariamente con un sistema uno-a-molti (un server, n client).

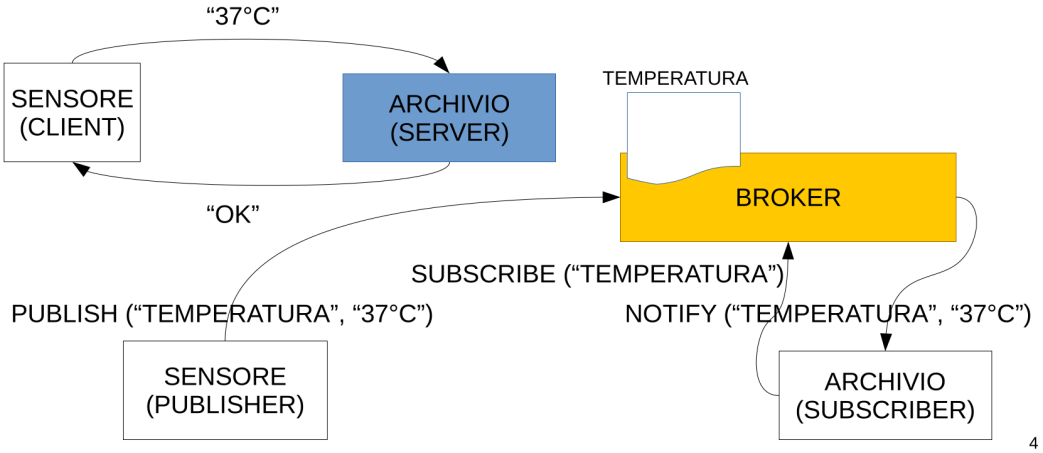
Svantaggi:

1. Complessità Aggiuntiva: l'introduzione di un Message Broker aggiunge un componente in più all'architettura, aumentando la complessità operativa (installazione, configurazione, monitoraggio del broker stesso);
2. Debugging: tracciare il flusso di un messaggio può essere più difficile a causa del disaccoppiamento e della natura asincrona. I messaggi non vanno da A a B direttamente, ma passano per un intermediario;
3. Latenza: in alcuni casi, l'introduzione del broker può aggiungere una piccola latenza rispetto alla comunicazione diretta (anche se broker moderni sono estremamente veloci).

Esempio:

Un esempio reale può essere un feed di notizie: un giornale (Publisher) pubblica articoli su vari topic (Sport, Politica, Tecnologia). Gli utenti (Subscriber) si iscrivono ai topic che li interessano e ricevono solo le notizie pertinenti.

Esempio di trasformazione (una domanda esame potrebbe essere di convertire un'applicazione client-server in pub/sub, quindi studiare e seguire questo modello di esempio):



4

Con il riquadro blu vediamo il modello classico client-server, mentre con il riquadro giallo possiamo vedere il modello pub/sub.

Alcuni esempi di protocolli (elenco puntato nero) e implementazioni specifiche dello stesso (elenco puntato bianco) sono:

- Message Queuing Telemetry Transport (MQTT):
  - Mosquitto (C/C++);
  - Paho (Python).
- Advanced Message Queuing Protocol (AMQP):
  - RabbitMQ;
- Apache Kafka: piattaforma distribuita di gestione di stream di eventi.

MQTT:

MQTT è un protocollo di messaggistica leggero, progettato per la comunicazione efficiente tra dispositivi, specialmente quelli con poche risorse o su reti inaffidabili, come nell'Internet of Things. Funziona sul modello Publisher/Subscriber come descritto. Ciò che lo rende potente sono funzionalità come i diversi livelli di qualità del servizio (QoS) per garantire la consegna, i messaggi "retained" per mostrare l'ultimo stato e il "Last Will and Testament" per notificare disconnessioni improvvise. È l'implementazione ideale del Pub/Sub per il mondo dell'IoT.

N.B: la parte vulnerabile è il broker!!!