Pub/Sub

Ashish Pratap Singh

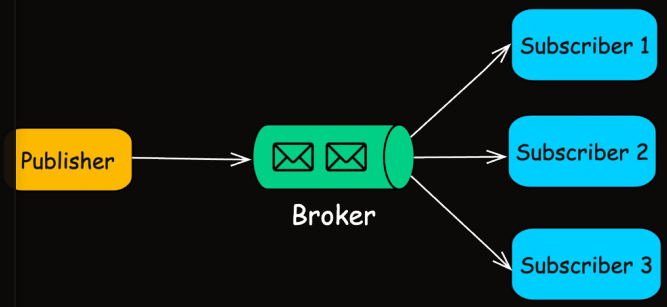
2 min read

ublish-Subscribe (Pub/Sub) is an architectural pattern sed to enable asynchronous communication between omponents in a distributed system.

n a Pub/Sub system, components can publish messages o a topic without needing to know who will receive them. imilarly, components can subscribe to topics to receive essages without needing to know the details of the ublishers.

his decoupling of publishers and subscribers offers sev- ral advantages, such as improved scalability, easier aintenance, and the ability to build more resilient sys- ems.

. Components of Pub/Sub



ublishers

ublishers are components or services that send mes- ages (or events) to a topic. They generate data or vents and "publish" these messages without worrying bout who will consume them.

xample: In an e-commerce system, a service that han- les order processing might publish an "Order Placed" vent whenever a new order is received.

ubscribers

Subscribers are components or services that express interest in certain topics. They receive and process messages that are published to those topics.

Example:A notification service might subscribe to the "Order Placed" topic to send confirmation emails to customers.

Message Broker

The **Message Broker** (or message queue) is the core component that manages topics, receives messages from publishers, and distributes them to all the subscribers of those topics. It acts as the intermediary to ensure that messages flow efficiently between publishers and subscribers.

Popular Message Brokers:

- Apache Kafka
- RabbitMQ
- Google Cloud Pub/Sub
- Amazon SNS/SQS

---

2. How Pub/Sub Works

Let's walk through a typical Pub/Sub interaction:

1. **Publishers Send Messages:** A publisher creates a message and sends it to a specific topic on the message broker.
2. **Message Broker Receives and Stores Messages:** The broker receives the message and temporarily stores it in the topic. Depending on the implementation, messages may be persisted to disk for durability.
3. **Subscribers Receive Messages:** Subscribers that have subscribed to the topic receive the message. This can happen in real time, or the subscriber may poll the broker if real-time delivery is not required.
4. **Processing and Acknowledgment:** Subscribers process the message. In some systems, the subscriber sends an acknowledgment back to the broker to confirm that the message has been processed successfully.

---

3. Benefits of Using Pub/Sub

- **Loose Coupling:** Publishers and subscribers do not need to know about each other. This makes it easier to develop, maintain, and scale each component independently.
- **Handling High Throughput:** A message broker can efficiently manage and route a large volume of messages, making it ideal for high-traffic systems.
- **Asynchronous Processing:** Systems can process messages at their own pace, which helps to balance loads during peak times.
- **Easy Integration:** New subscribers can be added to a topic without modifying the publishers.

- **Real-Time Communication:** Enables real-time data dissemination, which is crucial for applications like live notifications, event tracking, and IoT data collection.
- **Message Durability:** Many brokers can persist messages to ensure they aren't lost, even if parts of the system fail.
- **Fault Tolerance:** The decoupled nature of Pub/Sub can help isolate and manage failures more effectively.

---

4. Common Use Cases

Real-Time Notifications

- **Example:** Social media platforms using Pub/Sub to deliver instant notifications about likes, comments, or messages.

Event Logging and Analytics

- **Example:** Systems like Apache Kafka are used to stream logs from various services for real-time analytics and monitoring.

Microservices Communication

- **Example:** In a microservices architecture, services can communicate asynchronously using Pub/Sub, enabling scalable and resilient workflows.

IoT Data Aggregation

- **Example:** Devices in an IoT network publish sensor data to topics, which can then be aggregated and processed in real time.