WSO2CON 2024

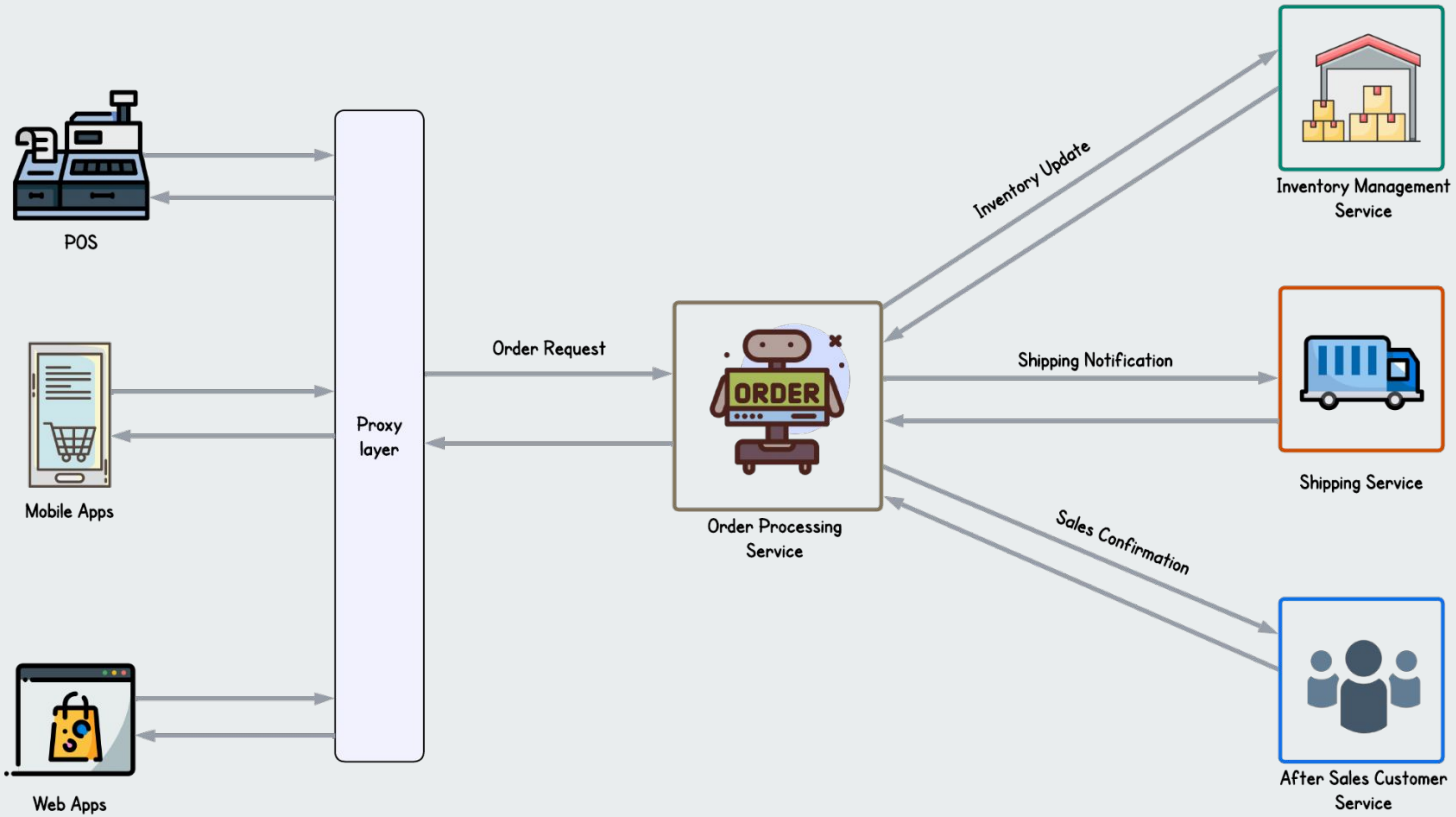
Designing Event-Driven Enterprises



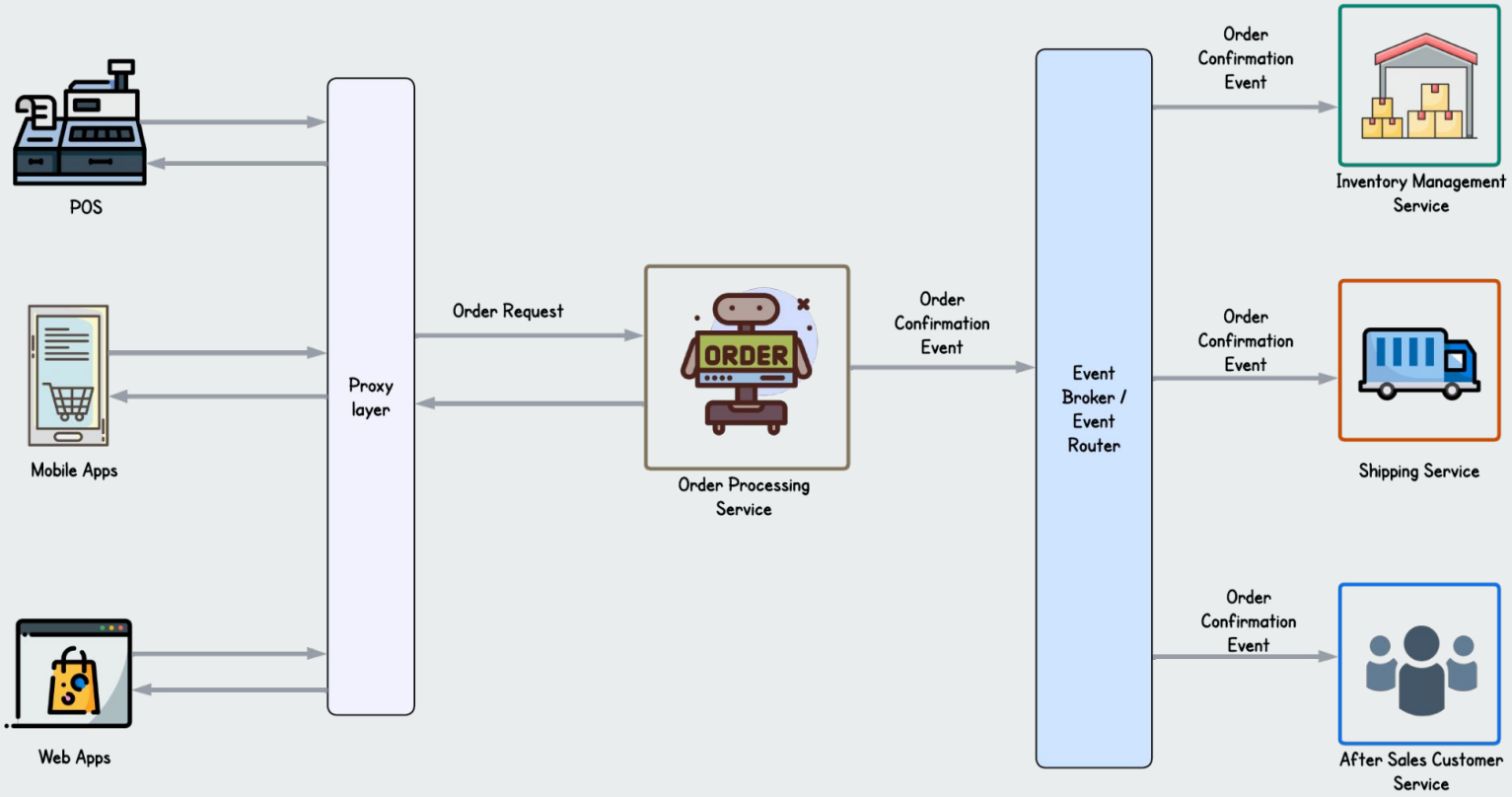
Mohanadarshan Vivekanandalingam
Associate Director / Architect
WSO2

Designing Event-Driven Enterprises : Stories of Transformation

Reference Architecture : Retail & eCommerce



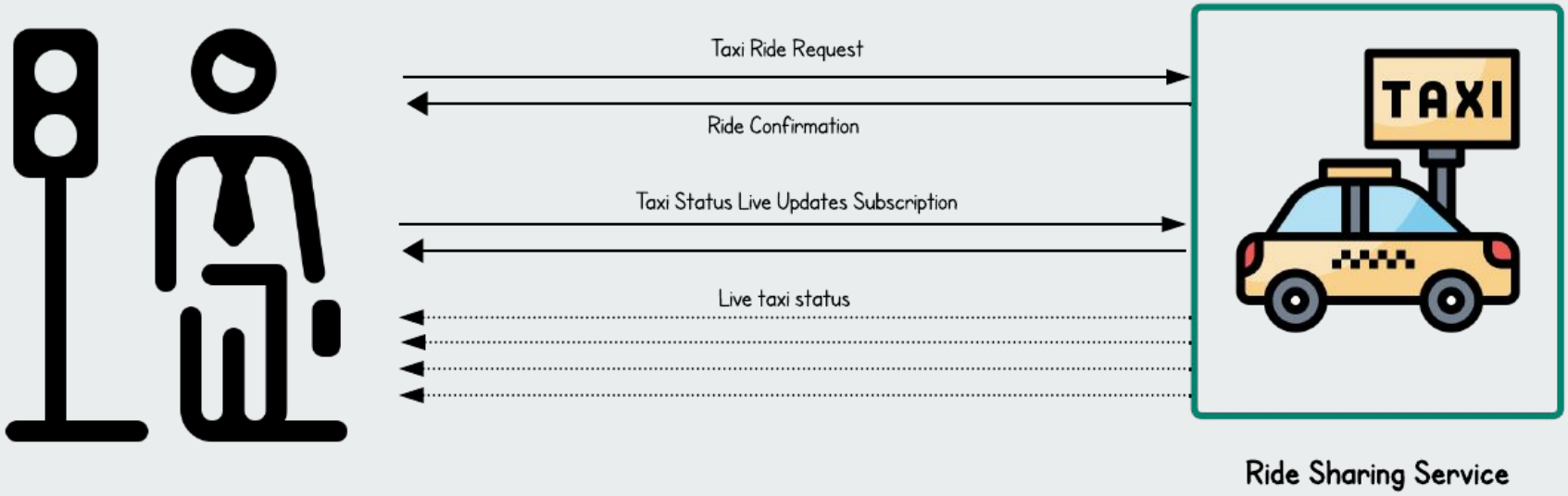
Event-Driven Reference Architecture : Retail & eCommerce



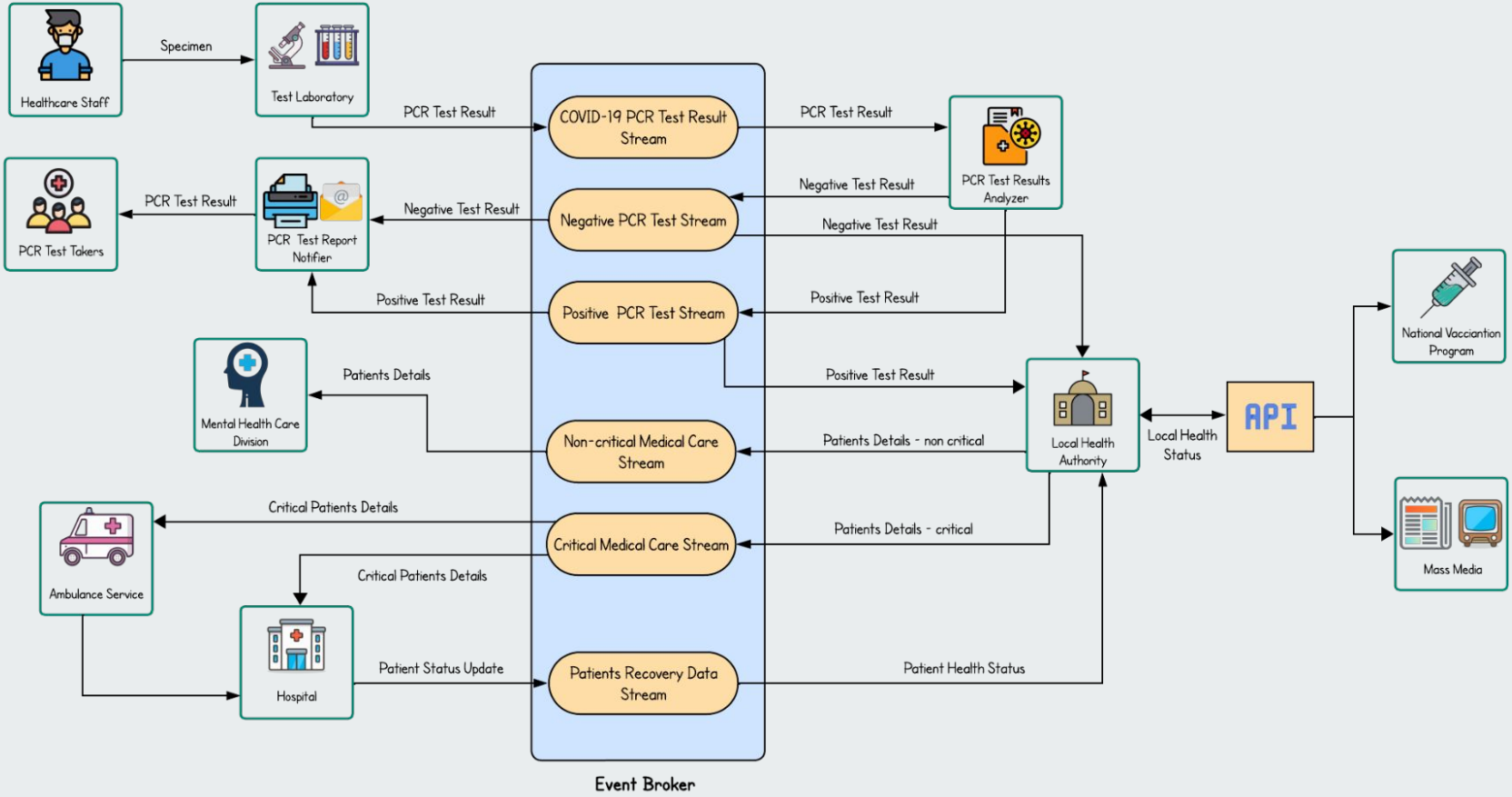
Reference Architecture : Ride Sharing App



Event-Driven Reference Architecture : Ride Sharing App



COVID-19 Task Force : Event – Driven Reference Architecture

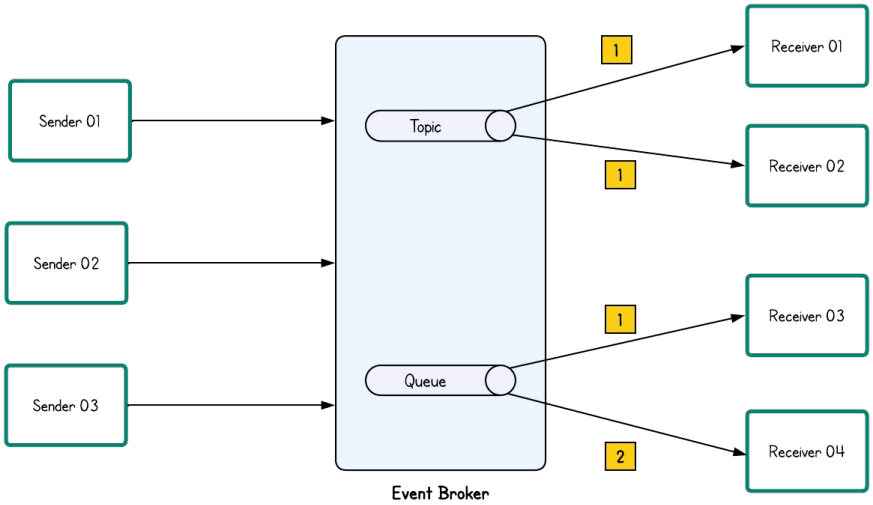


Key Concepts & Components

Components of Event-Driven Systems

- **Event** - A Significant change in state, which is triggered when a user takes an action
- **Event Schema** - A structured representation of event.
- **Event Producer** - Something that generates or creates data points or events
- **Event Consumer/Handler** - Something that receives or utilizes data points or events generated by an event producer for analysis or processing
- **Event Broker** - A middleman that manages the flow of data between event producers and consumers.
- **Event Processing** - Handling and analyzing data as it happens or is generated in real-time, allowing for immediate insights

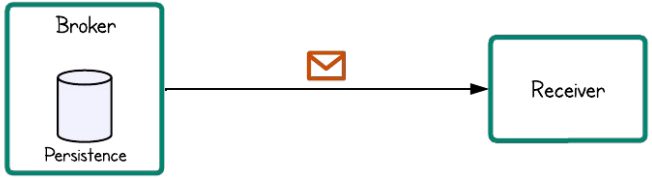
Event Broker



- Routes events between systems using the publish-subscribe messaging pattern.
- Introduce flexibility by decoupling publishers and subscribers of an application.
- Selecting a suitable event broker is a key in designing event-driven enterprises.
- Supports push and pull consumption models.
- Most event brokers support in-memory and file based message stores.

Event Broker – Quality of Service

Persistence



- Event loss is not acceptable even there is a failure in event broker or event handlers.

At least once delivery



- Multiple attempts can be made. Event duplicates are possible

Exactly once delivery



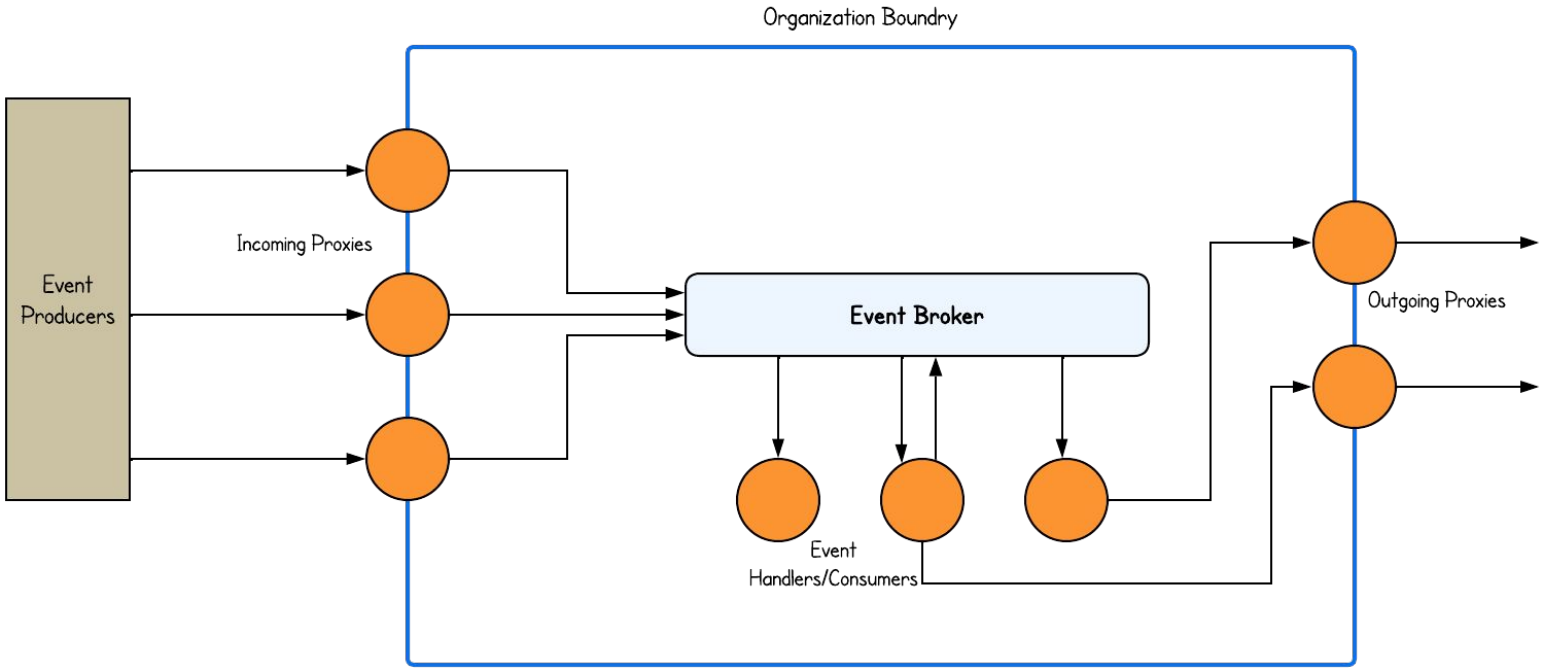
- Event cannot be lost or duplicated

In-order delivery



- Event order cannot change

Event-Driven Reference Architecture

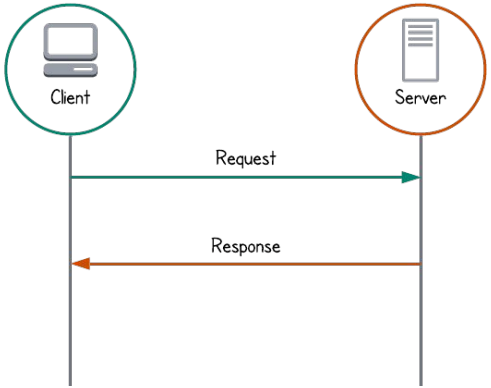


https://wso2.com/blogs/thesource/exploring-event-driven-architecture-a-beginners-guide-for-cloud-native-developers/ by Srinath Perera

Incoming/Outgoing Proxies

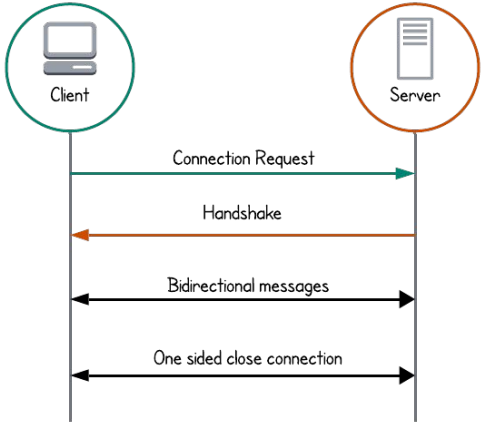
REST/JSON RPC

- Request/Response
- No Streaming Support
- Simple but limited support for real time communication



WebSocket

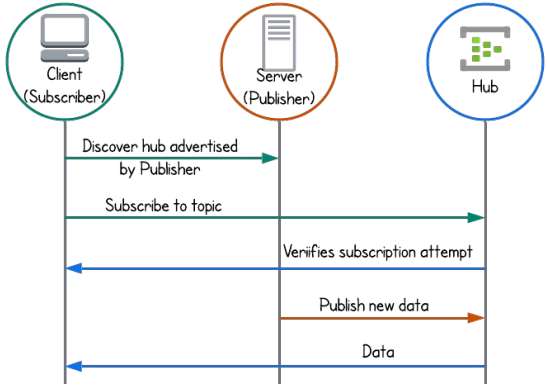
- Full-duplex
- Streaming Available
- Low latency | Requires persistent connections



Websub

Websub

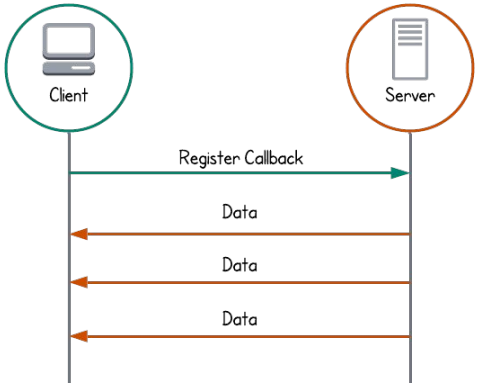
- Asynchronous
- Streaming Available
- Requires additional infrastructure



Webhook

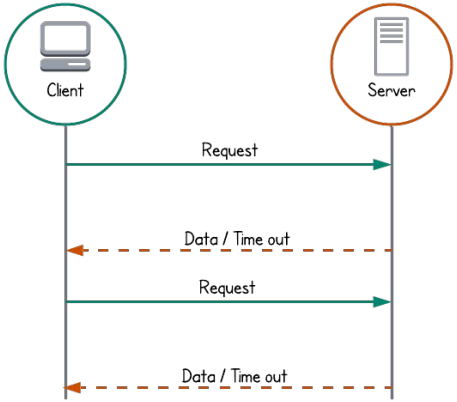


- Asynchronous | one way
- No Streaming Support
- Requires publicly accessible endpoints



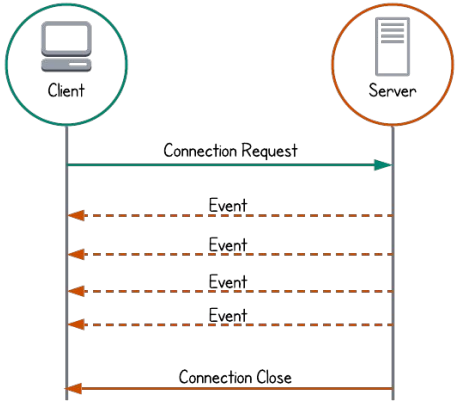
Polling Client

- Request/Response
- Streaming Available
- Easy to Implement | Universally supported
- High latency and high network load



SSE

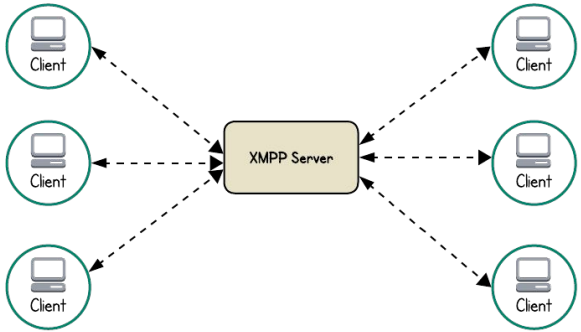
- Server initiated
- Streaming Available
- Simplified API for EDA
- Limitation in open connections | Does not support binary data



XMPP



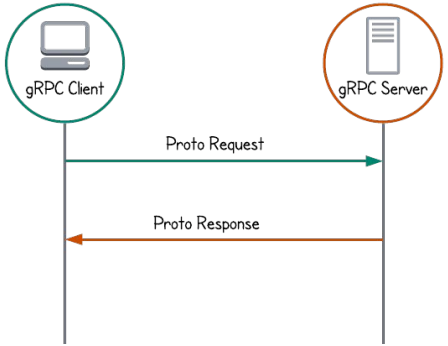
- Asynchronous
- Streaming Available
- Decentralized but complex protocol



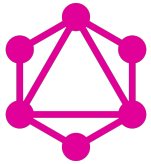
gRPC



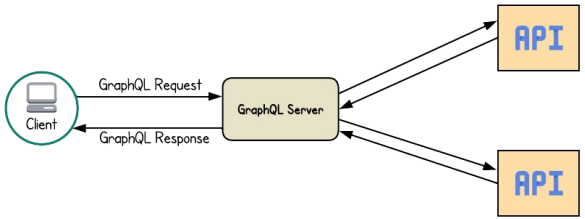
- Full-duplex
- Streaming Available
- Efficient but complex setup



GraphQL



- Query/Response
- Streaming Available
- Complex schema definition and learning curve



Event Processing

Common Types in Event Processing

Stateless event processing

- Filtering
- Event transformation
- Routing
- Event enrichment

Stateful event processing

- Scatter & Gather
- Aggregation
- Reorder Events
- Pattern & Sequence Detection
- Windowed event processing
- Rate Limiting

Decision Making : Event-Driven Systems

Benefits of Event-Driven Paradigm

- Scale Independently (essential in cloud-native deployments)
- Fault tolerance and fail Independently (Impact Isolation).
- Increase agility | Ease of adding services
- Resilient for event bursts
- Independently build and deploy applications
- Easy integration with heterogeneous systems



Image by freepik

Challenges in Event-Driven Solutions

- Complexity
- Debugging is not easy
- Monitoring difficulties
- Synchronization is not guaranteed
- Rollbacks are not easy
- No central orchestration
- Lack of skilled developers



Image by freepik

Common Mistakes in Event-Driven Designs

- Implementing Event-Driven Design universally!
- Disregarding the importance of Event Schema
- Attempting to replace necessity for an Event Broker with a Database
- Not evaluating the adoptability of the existing deployment architecture
- Insufficient grasp of the end-to-end use case
  - Need for a stateful processing
- Not designed to co-exist with legacy systems.
- Overthinking scalability and reusability! (Not only in EDA)



When to Choose Event-Driven Paradigm?

- Throughput takes precedence over latency
- Parallel processing needs in Integration
- Loose coupling is essential between components
- Realtime processing requirements
- Not the great choice for client facing use cases
- Eventual consistency is acceptable
- The benefits of EDA outweigh the drawbacks



Conclusion

- Event-Driven Designs empowers developers to build flexible, scalable, and responsive systems that can seamlessly integrate with external services
- Use Event-Driven Design wisely; don’t over use it. Don’t apply universally.
- Event Brokers are unavoidable in Event-Driven enterprises.
- Databases won’t replace Event Brokers, AND VICE VERSA
- Proficiency in designing event-driven systems is essential; invest in thorough gathering of requirements, rather than applying it directly.

Question Time!



Thank You!