Communication Patterns

Agenda

- Communication Patterns
- Synchronous Pattern
- Asynchronous Patterns
- Message Brokers: Rabbit MQ
- MassTransist

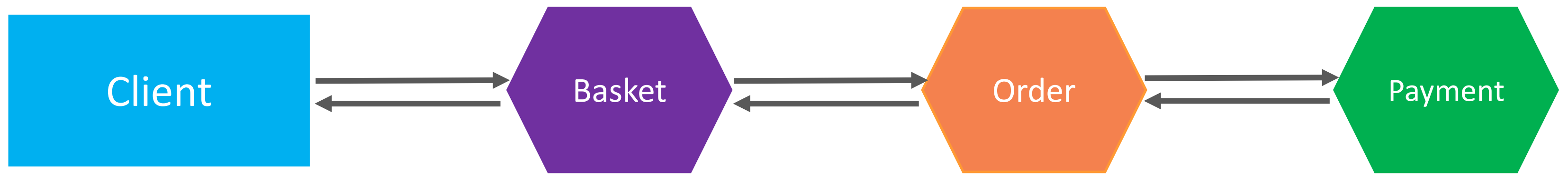
ScholarHat

Communication Patterns

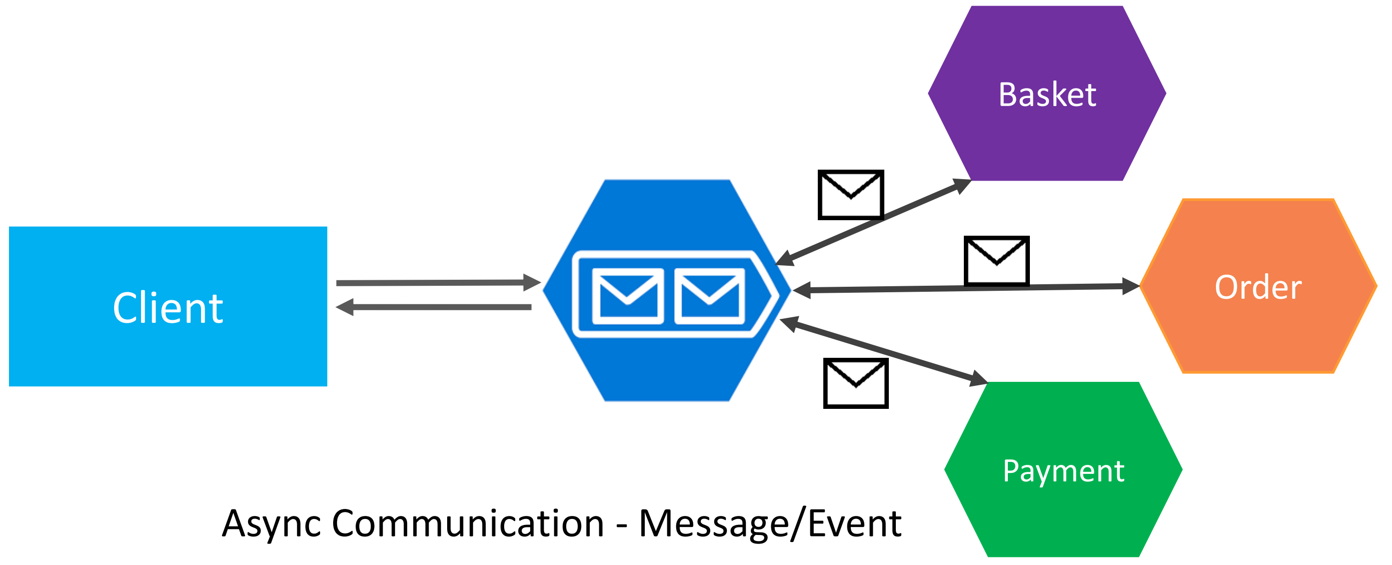
Problem: How do microservices communicate with each other?

- **Request/Reply** - HTTP/HTTPs is a synchronous protocol. The client sends a request and waits for a response from the service.
- **Messaging Pattern** - Services communicate with each other using messages. Sender services push messages to a message broker that other services subscribe to.
- **Event Driven Pattern** - In an event-driven approach the communication between services happens via events that individual service produce and the consuming services react to the occurrence of the event.

Sync and Async Communication

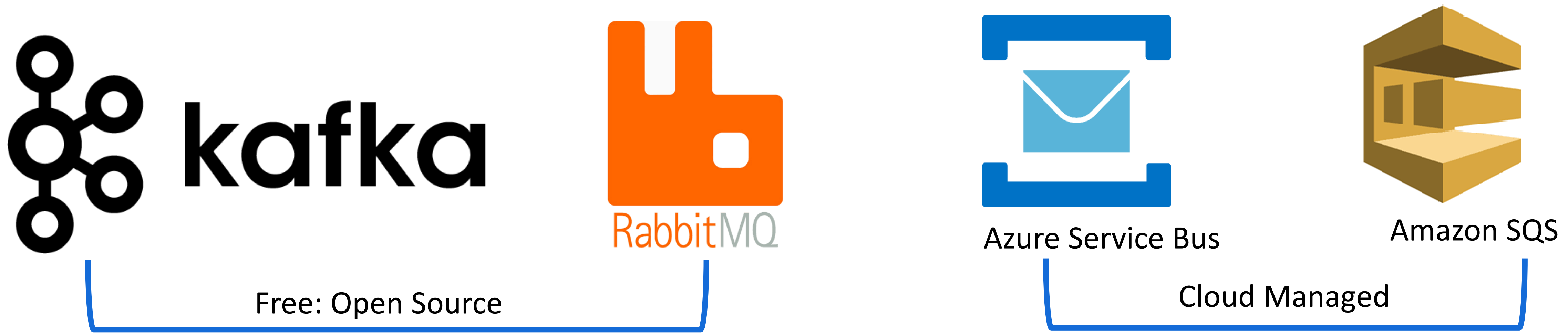


Sync Communication - Request/Reply



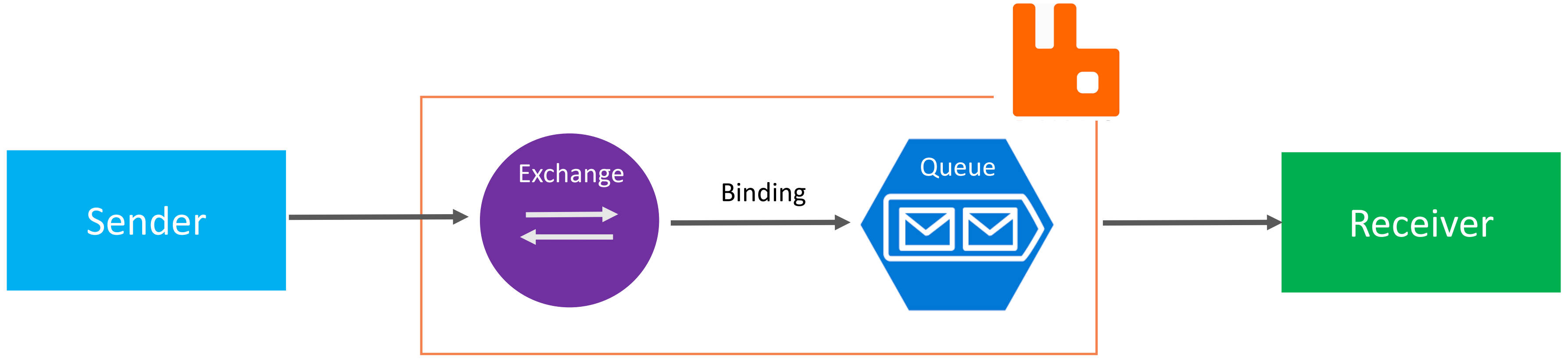
Async Communication - Message/Event

Message Brokers



RabbitMQ

- The most widely deployed open source message broker.
- Lightweight and easy to deploy on premises and in the cloud.
- Supports multiple messaging protocols like AMQP, STOMP, MQTT etc.



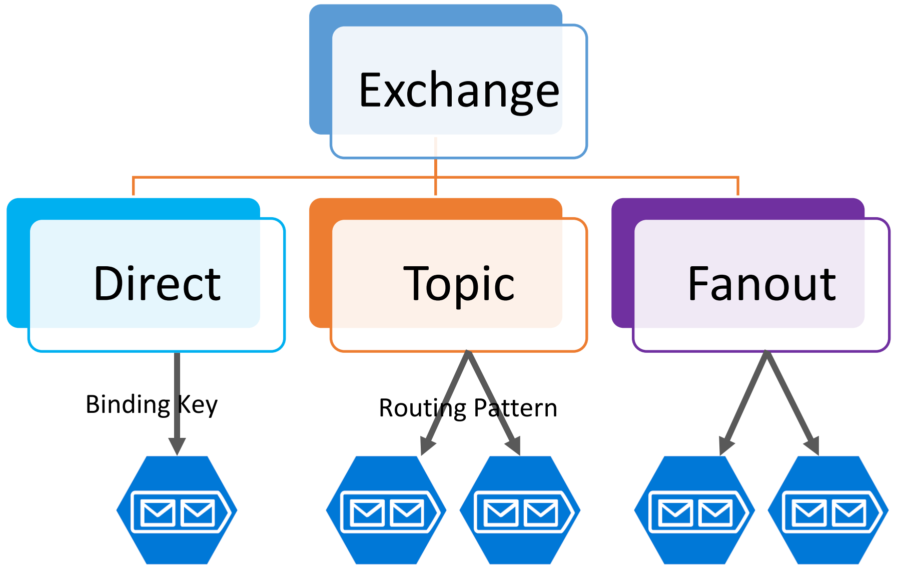
ScholarHat

RabbitMQ Components

- **Queue** - A queue is a buffer that stores messages for consumers to retrieve. A queue is bound to one or more exchanges, occasionally with a routing key. A queue may have one or many consumers.
- **Message** - A message is what is transported between the publisher and the consumer, it's essentially a byte array with some headers on top.
- **Exchange** - Receives messages from producers and deliver them to queues depending on rules defined by the exchange type. In order to receive messages, a queue needs to be bound to at least one exchange.
- **Binding** - A binding is a link between a queue and an exchange. It determines to which queues a message should be routed and it may be to zero or many.
- **Routing key** - A part of the header of every message and used to route the message.
- **Connection -** A TCP connection between your application and the RabbitMQ broker.
- **Channel -** A virtual connection inside a connection. When publishing or consuming messages from a queue - it's all done over a channel.

Types of Exchange

- **Direct**: The message is routed to the queues whose binding key exactly matches the routing key of the message. **For example**, if a queue is bound to the exchange with the **binding key pdfprocess**, a message published to the exchange with a **routing key pdfprocess** is routed to that queue.
- **Topic**: The topic exchange does a wildcard match between the routing key and the routing pattern specified in the binding.
- **Fanout**: A fanout exchange routes messages to all of the queues bound to it. The keys provided will simply be ignored.



Exchange Pre-declared Names

Exchange type	Default pre-declared names
Direct exchange	(Empty string) and amq.direct
Fanout exchange	amq.fanout
Topic exchange	amq.topic
Headers exchange	amq.match (and amq.headers in RabbitMQ)

RabbitMQ Docker Setup

- docker run -p 15672:15672 -p 5672:5672 --name rabbitmq rabbitmq:3-management
- Access UI: http://localhost:15672/
- Login: UserId: guest, pwd: guest
- WSL Setup In Step 4: Manual installation steps for older versions of WSL | Microsoft Docs

RabbitMQ Machine Setup

- Install the Erlang and RabbitMQ
- https://www.erlang.org/downloads
- https://www.rabbitmq.com/install-windows.html#installer
- Enable Plugins: https://www.rabbitmq.com/management.html
- *> rabbitmq-plugins enable rabbitmq_management*
- > rabbitmqctl status
- Setup Video: https://www.youtube.com/watch?v=UnKbvqVKB7k
- Access UI: http://localhost:15672/
- Login: UserId: guest, pwd: guest

ScholarHat

MassTransit

- A free, open-source, lightweight message bus used to create distributed applications using .NET technologies.
- An abstraction between the message brokers and the application.
- Supports Rabbitmq, Azure Service Bus, and Amazon SQS/SNS etc.
- Supports message patterns such as retry, circuit breaker, outbox.
- Support for distributed transaction using Saga, event-driven state machines.
- Built-In exception Handling.

MassTransit Methods

- **Publish()** - Useful to **send an event** since an event can be observed by one or more listeners. Follows the Publish-Subscribe pattern.
- When a message is published, it is not sent to a specific endpoint, but broadcasted to any consumers which have subscribed to the message type.
- **Send()** - Useful to **send a command** since a command needs to be executed once. Need to specify the endpoint i.e. which queue to go to use send method.
- When a message is sent, it is delivered to a specific endpoint using a *DestinationAddress*.

Key Points About MassTransit

- MassTransit creates durable, fanout exchanges by default, and queues are also durable by default.
- MassTransit encourages and support the use of interfaces for message contracts, and initializers make it easy to produce interface messages.

ScholarHat