Chapter 06 IPC

Overview of IPC in Microservices

	one-to-one	one-to-many
Synchronous	Request/response	—
Asynchronous	Asynchronous request/response One-way notifications	Publish/subscribe Publish/async responses

- **Interaction style**
  - **One-to-One, One-to-Many**
  - **Synchronous** >> Request-response based communication (REST, gRPC)
    - อาจมีการ Block เพื่อรอ response
  - **Asynchronous** >> Message-based communication (AMQP, STOMP)
- **Mesage** เป็นได้ทั้ง
  1. **Human readable** >> JSON, XML
  2. **Binary** >> Protocol Buffer

> 💡 สามารถออกแบบ API ได้โดยใช้ **Interface Definition Language (IDL)**

Synchronous RPI Pattern

- **Remote Invocation Pattern (RPI)** เป็น pattern ที่ client จะแลกเปลี่ยน request response กับ server โดย client เชื่อว่า จะได้รับ response กลับมาในเวลาไม่นาน
  - some client ทำการ **blocking** เพื่อรอ response และใช้ **non-blocking architecture** เพื่อให้ reactive
- **REST**
  - ข้อดี
    1. ง่าย ไม่จำเป็นต้องใช้ **Intermediate Broker**
    2. จากที่ใช้ HTTP protocol ทำให้ firewall friendly
  - ข้อเสีย
    1. Support แค่ request-response styles
    2. Availability ต่ำ
    3. Client จำต้องรู้ static IP ของ service instance (ยกเว้นแต่จะใช้ **Service Discovery**)
    4. fetch multiple resource ในหนึ่ง request เป็นเรื่องที่ยาก
- **gRPC**
  - ข้อดี
    1. Compact ส่ง large data ได้ดี
    2. มี Bidirectional streaming
    3. เขียนได้ด้วยหลายภาษา
  - ข้อเสีย
    1. ถ้าเขียนด้วย JS จะยุ่งยากกว่า REST นิดหน่อย
    2. firewall เก่า ๆ อาจไม่ support HTTP/2

💡 REST เขียนด้วย IDL
gRPC เขียนด้วย Proto-based IDL

Circuit Braker Pattern

- **Synchronous** อาจทำให้เกิด partition failure >> **API Gateway** ต้องป้องกันไม่ให้เกิด
- วิธีการแก้ปัญหา
  1. ออกแบบ **RPI Proxy** ให้ handle unresponsive ได้
     a. Network Timeout >> Block แค่ถึงเวลา ๆ หนึ่ง
     b. Limit number of requests from client >> จำกัดจำนวน request ที่ยิงได้ ณ ขณะหนึ่งหน่วยเวลา
     c. Circuit Breaker Pattern >> พิจารณาจาก success/failed request ถ้า error rate สูงกว่าที่กำหนดไว้ ปัดตก request ทันที
  2. ตัดสินใจว่าจะ recover from failure อย่างไร
     a. Return error ให้กับ client
     b. Return fallback value >> ประมาณว่าเป็นรหัส code ว่าเกิดปัญหา case นี้ขึ้น

Service Discovery

จากการที่ Service มี Dynamic IP address และ Port

- **Service Registry** >> Database ที่รวม network location ของ services
  - มีการ update เรื่อย ๆ ว่า instance ไหน start/stop อยู่
- Implement ได้ 2 วิธี

(1) Application-Level Service Discovery

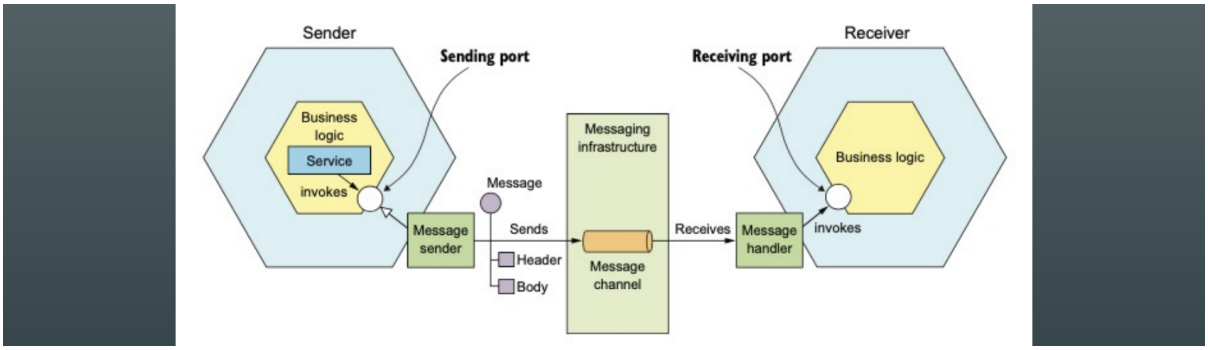
- **Self registration** >> service ไหนเพิ่มเข้ามาหรือหยุด ต้องส่ง API ไปบอก Service Registry
- **Client-side discovery** >> ที่ client มี Service Discovery Library เขียนไว้อยู่

(2) Platform-Provided Service Discovery

- **3rd-party registration** >> Registar ทำหน้าที่ observe service และ update service registry ให้
- **Service-side discovery** >> Platform Router บอก network address ของ service ให้กับ client

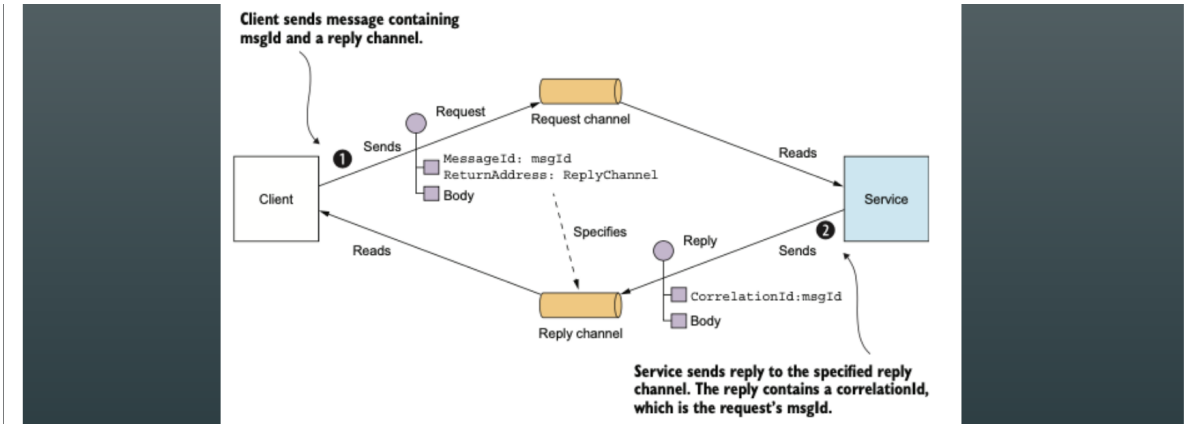
Asynchronous Pattern

- เป็น pattern ที่ client จะแลกเปลี่ยน **message** กับ server ผ่านทาง **channel**
  - เนื่องจากไม่ได้คาดหวังว่าจะได้รับการตอบกลับมา ดังนั้นจะไม่ Block เพื่อรอ reply



- **message** แบ่งออกเป็น 3 ประเภทคือ Document, Command และ Event

- **channel** แบ่งออกเป็น 2 ประเภทคือ
  1. **Point-to-Point channel** >> deliver to one consumer
  2. **Publish subscribe** >> deliver to all consumer
- **Interaction Style** เริ่มต้นที่เวลาส่ง ส่งไปทั้ง **message ID** และ **Reply Channel** เพื่อที่หากมีการตอบกลับแล้วจะได้ส่งกลับมาถูก channel



Brokerless && Broker-Based Architecture

Brokerless

- ข้อดี
  - Lower Latency and Operational Complexity
  - Lower chance of Bottle Neck and Single Point of Failure
- ข้อเสีย
  - ต้องจัดการ Service Discovery เอง
  - Lower Availability >> จากการที่ทั้งสองฝั่งต้อง available เพื่อแลกเปลี่ยน message

Broker-based >> เป็นตัวกลางในการแลกเปลี่ยน message

- ข้อดี
  - Loose couple >> sender ไม่ต้องรู้ location ของ consumer ตัว message broker จัดการให้
  - Message Buffering >> Message Broker จัดการ buffer ไว้จนกว่าง consumer จะพร้อม
  - Flexible Communication
- ข้อเสีย
  - Potential Bottle Neck and Single Point of Failure
  - Additional Operational Complexity

Message Ordering

- เป็นหนึ่งในปัญหาที่อาจเกิดหากใช้ **Brokerless**
- แก้ได้ด้วย **shard partition channel** และ **shard key** (shard ลำดับต่ำให้ทำก่อน)

Duplicate Message

- เป็นหนึ่งในปัญหาที่อาจเกิดหากใช้ **Brokerless**
- ไม่อยากให้เกิด duplicate message >> track ด้วย **message ID** (ถ้าทำไปแล้วไม่ทำซ้ำ)