**🚀 Individual Examples for Each Pattern**

**1. API Gateway Pattern**

* **What**: A gateway that routes client requests to backend microservices.
* **How**: Use **Spring Cloud Gateway**.
* **Execution**:
* mvn spring-boot:run -pl api-gateway
* **Example**:
  + Request: http://localhost:8080/products
  + Gateway forwards to Product Service (http://localhost:9001/products).

**2. Aggregator Pattern**

* **What**: Combines results from multiple services into a single response.
* **How**: Order Service calls Product Service + Inventory Service.
* **Execution**:
  + Run Product + Inventory + Order Service.
* **Example**:
  + GET http://localhost:9004/order-summary/1
  + Response contains product details + stock info.

**3. Proxy / Adapter Pattern**

* **What**: A microservice acts as an adapter to an external API.
* **How**: Payment Service wraps a fake “Bank API”.
* **Execution**:
  + Call Order → internally calls Payment (adapter).
* **Example**:
  + POST /orders triggers Payment Adapter → returns success/failure.

**4. Chained Microservice Pattern**

* **What**: A request passes through multiple services sequentially.
* **How**: Order → Payment → Notification.
* **Execution**:
  + Place order → Payment processed → Notification logged.
* **Example**:
  + Console shows:  
    Order Created → Payment Success → Notification Sent.

**5. Event-Driven Pattern (Pub/Sub)**

* **What**: Services communicate asynchronously via events.
* **How**: Spring ApplicationEventPublisher.
* **Execution**:
  + Order Service publishes OrderCompletedEvent.
  + Notification Service subscribes.
* **Example**:
  + Place order → logs in Notification Service: 📩 Notification sent.

**6. CQRS (Command Query Responsibility Segregation)**

* **What**: Separate command (write) and query (read) models.
* **How**: OrderController has two endpoints:
  + POST /orders → Command (create order).
  + GET /orders/{id} → Query (check order status).
* **Execution**:
  + Place an order → query order status.

**7. Saga Pattern**

* **What**: Long transaction split into smaller steps with rollback.
* **How**: If Payment fails, Order is cancelled (compensating transaction).
* **Execution**:
  + Try placing order with amount > 100 → Payment fails → Order cancelled.
* **Example**:
  + Console logs: Payment failed → Cancelling Order.

**8. Database per Service (H2 DB)**

* **What**: Each microservice has its own database.
* **How**: ProductService has productdb; OrderService has orderdb.
* **Execution**:
  + Run both → Each has independent H2 console at /h2-console.
* **Example**:
  + Products stored in productdb, Orders stored in orderdb.

**9. Log Aggregation & Distributed Tracing**

* **What**: Trace a request across services.
* **How**: Spring Cloud Sleuth + Zipkin.
* **Execution**:
  + Run Zipkin: docker run -d -p 9411:9411 openzipkin/zipkin.
  + Call Order → see trace in Zipkin UI.
* **Example**:
  + Visit <http://localhost:9411> → traces visible.

**10. Service Discovery**

* **What**: Services register and discover dynamically.
* **How**: Eureka Discovery Server.
* **Execution**:
  + Run Eureka (port 8761).
  + Start Product + Order → they appear in Eureka dashboard.
* **Example**:
  + Visit <http://localhost:8761>.

**📊 Pattern → Example Mapping**

| **Pattern** | **Example Service(s)** |
| --- | --- |
| **API Gateway** | API-Gateway routes to ProductService |
| **Aggregator** | OrderService combines Product + Inventory |
| **Proxy / Adapter** | PaymentService adapts external Bank API |
| **Chained Microservice** | Order → Payment → Notification |
| **Event-Driven (Pub/Sub)** | Order publishes event, Notification listens |
| **CQRS** | OrderService (separate POST & GET endpoints) |
| **Saga** | Order rollback when Payment fails |
| **Database per Service (H2)** | ProductService DB vs OrderService DB |
| **Log Aggregation & Tracing** | Sleuth + Zipkin traces across services |
| **Service Discovery** | Eureka Server + registered microservices |

 api-gateway-pattern — Spring Cloud Gateway example (port **8080**)

 aggregator-pattern — Aggregator calling public APIs (port **9001**)

 proxy-adapter-pattern — Payment Adapter simulation (port **9002**)

 chained-pattern — Chained calls example (port **9004**)

 event-driven-pattern — Spring ApplicationEvent pub/sub (port **9005**)

 cqrs-pattern — Simple command/query separation with H2 (port **9006**)

 saga-pattern — Simple saga with compensation (port **9007**)

 db-per-service-pattern — H2-backed service (port **9008**)

 log-tracing-pattern — Sleuth-enabled service (port **9009**)

 service-discovery-pattern — Eureka server (port **8761**)