# 🚀 ADVANN – Phase 2

**Inventory + Order Lifecycle (Microservices Architecture)**

---

## 🎯 Phase 2 Objective

Implement a production-style:

- Inventory management
- Order lifecycle
- Payment integration
- Distributed compensation logic
- Inter-service communication using OpenFeign

---

## 🏗 Architecture Overview

Microservices involved:

```
product-service  → manages products + inventory
cart-service     → manages cart
order-service    → manages order lifecycle
payment-service  → manages Razorpay integration
```

Communication:

- order-service → product-service (reserve / confirm / release stock)
- payment-service → order-service (update payment status)

---

## 1️⃣ Inventory Lifecycle (product-service)

---

### 📦 Product Entity (Updated)

```
@Entity
@Table(name = "products")
@Data
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private BigDecimal price;

    @Column(nullable = false)
    private Integer stock;

    @Column(nullable = false)
    private Integer reservedStock = 0;

    @Version
    private Long version;
}
```

---

## 🔁 Inventory Lifecycle Rules

| Stage | Action |
|---|---|
| Order Placed | reserveStock() |
| Payment Success | confirmStock() |
| Payment Failed | releaseStock() |
| Cancel Before Confirm | releaseStock() |

---

## 🧠 Inventory Methods

```
@Transactional
public void reserveStock(Long productId, Integer quantity) {
    Product product = productRepository.findById(productId)
            .orElseThrow(() -> new RuntimeException("Product not found"));

    if (product.getStock() < quantity) {
        throw new IllegalStateException("Insufficient stock");
    }

    product.setStock(product.getStock() - quantity);
    product.setReservedStock(product.getReservedStock() + quantity);

    productRepository.save(product);
}

@Transactional
public void confirmStock(Long productId, Integer quantity) {
    Product product = productRepository.findById(productId)
            .orElseThrow(() -> new RuntimeException("Product not found"));
    product.setReservedStock(product.getReservedStock() - quantity);

    productRepository.save(product);
}

@Transactional
public void releaseStock(Long productId, Integer quantity) {
    Product product = productRepository.findById(productId)
            .orElseThrow(() -> new RuntimeException("Product not found"));

    product.setReservedStock(product.getReservedStock() - quantity);
    product.setStock(product.getStock() + quantity);

    productRepository.save(product);
}
```

---

### 🔐 Internal Inventory APIs

```
@RestController
@RequestMapping("/internal/products")
@RequiredArgsConstructor
public class ProductInternalController {

    private final ProductService productService;

    @PostMapping("/{productId}/reserve")
    public ResponseEntity<ApiResponse<Void>> reserveStock(
            @PathVariable Long productId,
            @RequestParam Integer quantity) {
        productService.reserveStock(productId, quantity);
        return ResponseEntity.ok(new ApiResponse<>(true, "Stock reserved", null));
    }

    @PostMapping("/{productId}/confirm")
    public ResponseEntity<ApiResponse<Void>> confirmStock(
            @PathVariable Long productId,
            @RequestParam Integer quantity) {
        productService.confirmStock(productId, quantity);
        return ResponseEntity.ok(new ApiResponse<>(true, "Stock confirmed", null));
    }

    @PostMapping("/{productId}/release")
    public ResponseEntity<ApiResponse<Void>> releaseStock(
            @PathVariable Long productId,
        @RequestParam Integer quantity) {
        productService.releaseStock(productId, quantity);
        return ResponseEntity.ok(new ApiResponse<>(true, "Stock released", null));
    }
}
```

---

## 2️⃣ Order Lifecycle (order-service)

---

### 📦 Order Entity

```
@Entity
@Table(name = "orders")
@Data
@Builder
public class Order {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private Long userId;

    @Enumerated(EnumType.STRING)
    private OrderStatus orderStatus;

    @Enumerated(EnumType.STRING)
    private PaymentStatus paymentStatus;

    private BigDecimal totalAmount;

    @OneToMany(mappedBy = "order", cascade = CascadeType.ALL)
    private List<OrderItem> orderItems;

    @Version
    private Long version;
}
```

---

### 📦 OrderStatus

```
public enum OrderStatus {
    CREATED,
    CONFIRMED,
    SHIPPED,
    DELIVERED,
    CANCELLED
}
```

---

### 📦 PaymentStatus

```
public enum PaymentStatus {
    PENDING,
    PAID,
    FAILED
}
```

---

### 🔗 Feign Client (order → product)

```
@FeignClient(name = "product-service")
public interface ProductClient {

    @PostMapping("/internal/products/{productId}/reserve")
    ApiResponse<Void> reserveStock(@PathVariable Long productId,
                                   @RequestParam Integer quantity);

    @PostMapping("/internal/products/{productId}/confirm")
    ApiResponse<Void> confirmStock(@PathVariable Long productId,
                                   @RequestParam Integer quantity);

    @PostMapping("/internal/products/{productId}/release")
    ApiResponse<Void> releaseStock(@PathVariable Long productId,
                                   @RequestParam Integer quantity);
}
```

---

### 🔥 Distributed-Safe placeOrder()

```
@Override
@Transactional
public OrderResponseDto placeOrder(Long userId) {

    CartResponseDto cart = cartClient.getCartByUserId(userId).getData();

    Order order = Order.builder()
            .userId(userId)
            .orderStatus(OrderStatus.CREATED)
            .paymentStatus(PaymentStatus.PENDING)
            .totalAmount(cart.getGrandTotal())
            .build();

    List<CartItemResponseDto> items = cart.getItems();

    try {

        for (CartItemResponseDto item : items) {
            productClient.reserveStock(item.getProductId(), item.getQuantity());
        }

        order = orderRepository.save(order);

        for (CartItemResponseDto item : items) {
            orderItemRepository.save(
                    OrderItem.builder()
                            .order(order)
                            .productId(item.getProductId())
                            .quantity(item.getQuantity())
                            .price(item.getPrice())
                            .build()
            );
        }

        cartClient.clearCart(userId);

        return getOrderById(order.getId());

    } catch (Exception e) {

        for (CartItemResponseDto item : items) {
            productClient.releaseStock(item.getProductId(), item.getQuantity());
        }

        throw new RuntimeException("Order placement failed", e);
    }
}
```

---

### 💳 Payment Update Logic

```
if (newStatus == PaymentStatus.PAID) {

    for (OrderItem item : orderItems) {
        productClient.confirmStock(item.getProductId(), item.getQuantity());
    }

    order.setPaymentStatus(PaymentStatus.PAID);
    order.setOrderStatus(OrderStatus.CONFIRMED);
}
else if (newStatus == PaymentStatus.FAILED) {

    for (OrderItem item : orderItems) {
        productClient.releaseStock(item.getProductId(), item.getQuantity());
    }

    order.setPaymentStatus(PaymentStatus.FAILED);
    order.setOrderStatus(OrderStatus.CANCELLED);
}
```

---

## 3 Payment Service (Razorpay)

---

### Create Razorpay Order

```
JSONObject orderRequest = new JSONObject();
orderRequest.put("amount", dto.getAmount().multiply(BigDecimal.valueOf(100)));
orderRequest.put("currency", "INR");
orderRequest.put("receipt", "receipt_" + dto.getOrderId());

Order razorpayOrder = razorpayClient.orders.create(orderRequest);
```

---

### Verify Payment

```
if (!generatedSignature.equals(dto.getRazorpaySignature())) {

    payment.setPaymentStatus(PaymentStatus.FAILED);
    paymentRepository.save(payment);

    orderClient.updatePaymentStatus(
            dto.getOrderId(),
            PaymentStatusUpdateRequestDto.builder()
                    .paymentStatus(PaymentStatus.FAILED)
                    .build()
    );

    throw new RuntimeException("Signature verification failed");
}
```

---

## 🧠 What Phase 2 Achieved

✔ Inventory lifecycle with reserved stock
✔ Order lifecycle with state control
✔ Payment integration with Razorpay
✔ Distributed compensation (basic Saga)
✔ Inter-service communication using Feign
✔ Optimistic locking
✔ Clean microservice boundaries

---

## 📌 Technical Concepts Covered

- Microservices architecture
- Optimistic locking
- Saga pattern (compensation style)
- Distributed consistency
- Payment verification
- Idempotency protection
- State machine validation