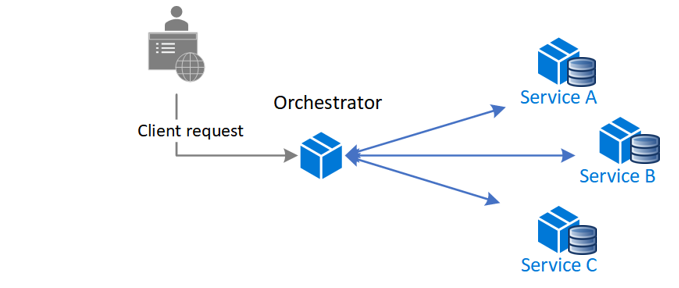
**Saga Orchestration Pattern with SpringBoot Kafka – 2022**

In case of Saga Orchestration Pattern, there is a Microservice called Orchestrator who manages all other services. Basically an Orchestrator is service with set of Listeners who listens and performs appropriate action for successful or failure scenarios. If the situation is not successful, it initiates a compensatory action. Each listener performs a specific task, basically each listener will make rest call to the respective microservice based upon the situation.



Let consider the following example.

**Order-Service**

@RestController

**public** **class** OrderController { 🡸 **Controller**

@Autowired

**private** OrderServiceImpl orderService;

@PostMapping(path = "/order")

**public** ResponseEntity<String> placeOrder(@RequestBody OrderEntity order) {

orderService.placeOrder(order);

String msg = "Order successfully placed ...";

**return** **new** ResponseEntity<String>(msg, HttpStatus.***CREATED***);

}

}

**Service Implementation**

@Service

**public** **class** OrderServiceImpl {

@Autowired

**private** KafkaTemplate<String, OrderEvent> kafkaTemplate;

@Value("${kafka.order.topic.name}")

**private** String orderTopicName;

**public** **void** placeOrder(OrderEntity order) {

OrderEvent orderEvent = **new** OrderEvent();

orderEvent.setOrderId(order.getOrderId());

orderEvent.setOrderName(order.getOrderName());

orderEvent.setActionName("ORDER\_PLACED");

orderEvent.setOrderPrice(order.getOrderPrice());

orderEvent.setShipToAddress(order.getShipToAddress());

// Publish

kafkaTemplate.send(orderTopicName, orderEvent);

System.***out***.println("Order sent to topic ...");

}

}

**Event**

@Data

**public** **class** OrderEvent {

**private** **long** orderId;

**private** String actionName;

**private** String orderName;

**private** **int** orderPrice;

**private** String shipToAddress;

}

**Configuration – application.properties**

server.port=8081

spring.application.name=saga-order-service

spring.profiles.active=dev

# Kafka Consumer

spring.kafka.consumer.bootstrap-servers=localhost:9092

spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer

spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer

spring.kafka.consumer.group-id=saga-order-grp-id

spring.kafka.consumer.auto-offset-reset=latest

# The below line is important for Json Deserialization

spring.kafka.consumer.properties.spring.json.trusted.packages=\*

#Kafka Producer

spring.kafka.producer.bootstrap-servers=localhost:9092

spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer

spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer

# Open API Swagger documentation

springdoc.swagger-ui.path=/index.html

springdoc.swagger-ui.disable-swagger-default-url=true

#Application Specific

kafka.order.topic.name=saga-choreo-order-topic

**Inventory-Service**

@RestController

**public** **class** InventoryController { 🡸 Controller

@Autowired

**private** InventoryServiceImpl inventoryService;

@PostMapping(path = "/inventory/item")

**public** ResponseEntity<String> processInventory(@RequestBody ItemRequestEntity itemRequest) {

System.***out***.println("Received request for : "+itemRequest);

inventoryService.initiatePayment(itemRequest);

**return** **new** ResponseEntity<String>("Item Received for processing", HttpStatus.***CREATED***);

}

}

**Service Implementation**

@Service

**public** **class** InventoryServiceImpl {

@Autowired

**private** KafkaTemplate<String, PaymentEvent> kafkaTemplate;

@Autowired

**private** KafkaTemplate<String, CancelEvent> cancelKafkaTemplate;

@Value("${kafka.inventory.out.topic.name}")

**private** String topicName;

@Value("${kafka.order.cancel.topic.name}")

**private** String cancelTopicName;

**private** **void** sendForPaymentInitiation(PaymentEvent paymentEvent) {

System.***out***.println("Initiating payment for the item");

kafkaTemplate.send(topicName, paymentEvent);

System.***out***.println("Sent for payment initiation ...");

}

**public** **void** initiatePayment(ItemRequestEntity itemRequest) {

**if**(itemRequest.getActionName().equalsIgnoreCase("ITEM\_AVAILABLE")) {

**boolean** itemAvlable = isItemAvailable(itemRequest.getItemName());

**if**(itemAvlable) {

PaymentEvent paymentEvent = **new** PaymentEvent();

paymentEvent.setOrderId(itemRequest.getOrderId());

paymentEvent.setItemName(itemRequest.getItemName());

paymentEvent.setPrice(itemRequest.getPrice());

paymentEvent.setActionName("INITIATE\_PAYMENT");

paymentEvent.setShipToAddress(itemRequest.getShipToAddress());

sendForPaymentInitiation(paymentEvent);

} **else** {

CancelEvent cancelEvent = **new** CancelEvent();

cancelEvent.setActionName("CANCEL\_ORDER");

cancelEvent.setReason("Outof Stock");

System.***out***.println("Order has been cancelled and item is put back in the inventory");

cancelKafkaTemplate.send(cancelTopicName, cancelEvent);

}

}

}

**private** **boolean** isItemAvailable(String name) {

**if**(name.startsWith("Vivo")) **return** **false**;

**return** **true**;

}

}

**Configuration – application.properties**

server.port=8082

spring.profiles.active=dev

# Kafka Consumer

spring.kafka.consumer.bootstrap-servers=localhost:9092

spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer

spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer

spring.kafka.consumer.group-id=saga-order-grp-id

spring.kafka.consumer.auto-offset-reset=latest

# The below line is important for Json Deserialization

spring.kafka.consumer.properties.spring.json.trusted.packages=\*

# Kafka Producer

spring.kafka.producer.bootstrap-servers=localhost:9092

spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer

spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer

# Open API Swagger Documentation

springdoc.swagger-ui.path=/index.html

springdoc.swagger-ui.disable-swagger-default-url=true

#Application Specific

kafka.inventory.out.topic.name=saga-inventory-out-topic

# To cancel the order

kafka.order.cancel.topic.name=saga-choreo-order-cancel-topic

**Payment Service**

@RestController

**public** **class** PaymentController { 🡸 Controller

@Autowired

**private** PaymentServiceImpl payService;

@PostMapping(path = "/payment")

**public** ResponseEntity<String> initiatePayment(@RequestBody PaymentRequest payRequest) {

System.out.println("Received request for payment: "+payRequest);

payService.processPaymentRequest(payRequest);

**return** **new** ResponseEntity<String>("Received Payment Request", HttpStatus.CREATED);

}

@PostMapping(path = "/undopayment")

**public** ResponseEntity<String> reversePayment(@RequestBody CancelRequest cancelRequest) {

System.out.println("Received request for payment: "+cancelRequest);

payService.reverseTransaction(cancelRequest);

**return** **new** ResponseEntity<String>("Received Payment Request", HttpStatus.CREATED);

}

}

**Service Implementation**

@Service

**public** **class** PaymentServiceImpl {

@Autowired

**private** KafkaTemplate<String, ShippingEvent> kafkaTemplate;

@Value("${kafka.shipping.out.topic.name}")

**private** String topicName;

@Autowired

**private** KafkaTemplate<String, CancelEvent> cancelKafkaTemplate;

@Value("${kafka.payment.cancel.topic.name}")

**private** String cancelTopicName;

**public** **void** reverseTransaction(CancelRequest cancelRequest) {

System.***out***.println("CancelRequest--->"+cancelRequest);

System.***out***.println("Your transaction has been reversed, you will get back your amount in 48 hours...");

}

**public** **void** processPaymentRequest(PaymentRequest payRequest) {

**if**(payRequest.getActionName().equals("RECEIVE\_PAYMENT")) {

**boolean** isReceived = isPaymentReceived(payRequest.getPrice());

**if** (isReceived) {

ShippingEvent shipEvent = **new** ShippingEvent();

shipEvent.setActionName("SHIP\_ITEM\_ADDRESS");

shipEvent.setItemName(payRequest.getItemName());

shipEvent.setOrderId(payRequest.getOrderId());

shipEvent.setShippingAdress(payRequest.getShippingAdress());

kafkaTemplate.send(topicName, shipEvent);

} **else** {

// Insufficient balance

CancelEvent cancelEvent = **new** CancelEvent();

cancelEvent.setOrderId(payRequest.getOrderId());

cancelEvent.setActionName("CANCEL\_ORDER");

cancelEvent.setReason("Insufficient fund...");

cancelEvent.setOrderName(payRequest.getItemName());

cancelKafkaTemplate.send(cancelTopicName, cancelEvent);

}

}

}

**public** **boolean** isPaymentReceived(**int** amount) {

**if** (amount > 50000)

**return** **false**;

**else**

**return** **true**;

}

}

**Configuration – application.properties**

server.port=8083

spring.profiles.active=dev

# Kafka Consumer

spring.kafka.consumer.bootstrap-servers=localhost:9092

spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer

spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer

spring.kafka.consumer.group-id=saga-order-grp-id

spring.kafka.consumer.auto-offset-reset=latest

# The below line is important for Json Deserialization

spring.kafka.consumer.properties.spring.json.trusted.packages=\*

# Kafka Producer

spring.kafka.producer.bootstrap-servers=localhost:9092

spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer

spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer

# Open API Swagger Documentation

springdoc.swagger-ui.path=/index.html

springdoc.swagger-ui.disable-swagger-default-url=true

#Application Specific Configuration

kafka.shipping.out.topic.name=saga-choreo-shipping-topic

# Listen from this topic

kafka.payment.cancel.topic.name=saga-choreo-payment-cancel-topic

**Shipping Service**

@RestController

**public** **class** ShippingController {

@Autowired

**private** ShippingServiceImpl shippingService;

@PostMapping("/ship")

**public** ResponseEntity<String> shipItem(@RequestBody ShipRequest shipRequest) {

System.out.println("ShipRequest---->"+shipRequest);

shippingService.shipItem(shipRequest);

**return** **new** ResponseEntity<String>("Package has been shipped...", HttpStatus.CREATED);

}

}

**Service Implementation**

@Service

**public** **class** ShippingServiceImpl {

@Autowired

**private** KafkaTemplate<String, OrderEvent> kafkaTemplate;

@Value("${kafka.order.complete.topic.name}")

**private** String topicName;

@Autowired

**private** KafkaTemplate<String, CancelEvent> cancelKafkaTemplate;

@Value("${kafka.cancel.ship.topic.name}")

**private** String cancelTopicName;

**public** **void** cancelOrder(CancelEvent cancelEvent) {

cancelKafkaTemplate.send(cancelTopicName, cancelEvent);

System.***out***.println("Your order processing has been cancelled due to invalid address");

}

**public** **void** shipItem(ShipRequest shipRequest) {

**boolean** invalidShipAdres = shipRequest.getShippingAdress().startsWith("Invalid");

System.***out***.println("invalidShipAdres : " + invalidShipAdres);

**boolean** validAction = shipRequest.getActionName().equalsIgnoreCase("SHIPPED\_COMPLETE");

**if** (validAction && !invalidShipAdres) {

OrderEvent orderEvent = **new** OrderEvent();

orderEvent.setOrderId(shipRequest.getOrderId());

orderEvent.setActionName("ORDER\_COMPLETE");

kafkaTemplate.send(topicName, orderEvent);

System.***out***.println("Your order processing has been complete");

} **else** {

CancelEvent cancelEvent = **new** CancelEvent();

cancelEvent.setActionName("CANCEL\_PAYMENT");

cancelEvent.setOrderId(shipRequest.getOrderId());

cancelEvent.setOrderName(shipRequest.getItemName());

cancelEvent.setReason("This item does not ship to address: " + shipRequest.getShippingAdress());

cancelKafkaTemplate.send(cancelTopicName, cancelEvent);

System.***out***.println("Your order processing has been cancelled due to invalid address");

}

}

}

**Configuration – application.properties**

server.port=8084

spring.profiles.active=dev

# Kafka Consumer

spring.kafka.consumer.bootstrap-servers=localhost:9092

spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer

spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer

spring.kafka.consumer.group-id=saga-order-grp-id

spring.kafka.consumer.auto-offset-reset=latest

# The below line is important for Json Deserialization

spring.kafka.consumer.properties.spring.json.trusted.packages=\*

# Kafka Producer

spring.kafka.producer.bootstrap-servers=localhost:9092

spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer

spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer

# Kafka Swagger Documentation

springdoc.swagger-ui.path=/index.html

springdoc.swagger-ui.disable-swagger-default-url=true

#Application Specific Configuration

#kafka.shipping.out.topic.name=saga-choreo-shipping-topic

kafka.order.complete.topic.name=saga-choreo-order-complete-topic

kafka.cancel.ship.topic.name=saga-cancel-ship-topic

**Main Orchestrator Service**

Orchestrator mainly contains listeners to listen to messages published by other microservices to particular topic and listeners makes appropriate service calls to take action.

@Component

**public** **class** PlacedOrderListener { 🡸 Order Listener

@Autowired

**private** InventoryRequestClient inventoryClient;

@KafkaListener(topics = "${kafka.order.topic.name}",groupId = "saga-order-grp-id")

**public** **void** listenOrderPlaced(OrderEvent orderEvent) {

System.***out***.println("Order Details:" + orderEvent);

**if**(orderEvent.getActionName().equalsIgnoreCase("ORDER\_PLACED")) {

// Make a call to inventory service

ItemRequest itemRequest = **new** ItemRequest();

itemRequest.setOrderId(orderEvent.getOrderId());

itemRequest.setItemName(orderEvent.getOrderName());

itemRequest.setPrice(orderEvent.getOrderPrice());

itemRequest.setActionName("ITEM\_AVAILABLE");

itemRequest.setShipToAddress(orderEvent.getShipToAddress());

inventoryClient.makeInventoryCall(itemRequest);

}

}

}

@Component

**public** **class** InventoryListener { 🡸 Inventory Listener

@Autowired

**private** PaymentClient payClient;

@KafkaListener(topics = "${kafka.inventory.out.topic.name}")

**public** **void** receivePayment(PaymentEvent paymentEvent) {

**if** (paymentEvent.getActionName().equals("INITIATE\_PAYMENT")) {

PaymentRequest payRequest = **new** PaymentRequest();

payRequest.setActionName("RECEIVE\_PAYMENT");

payRequest.setItemName(paymentEvent.getItemName());

payRequest.setOrderId(paymentEvent.getOrderId());

payRequest.setPrice(paymentEvent.getPrice());

payRequest.setShippingAdress(paymentEvent.getShipToAddress());

payClient.makePaymentCall(payRequest);

}

}

}

@Component

**public** **class** ShippingListener { 🡸 Shipping Listener

@Autowired

**private** ShippingRequestClient shipClient;

@KafkaListener(topics = "${kafka.shipping.out.topic.name}")

**public** **void** receiveItemForShipping(ShippingEvent shipEvent) {

System.***out***.println("ShippingEvent---->"+shipEvent);

//Make a call to Shipping API

ShipRequest shipReq = **new** ShipRequest();

shipReq.setActionName("SHIPPED\_COMPLETE");

shipReq.setItemName(shipEvent.getItemName());

shipReq.setOrderId(shipEvent.getOrderId());

shipReq.setShippingAdress(shipEvent.getShippingAdress());

shipClient.shipProduct(shipReq);

}

}

@Component

**public** **class** OrderCompleteListener { 🡸 Order complete listener

@KafkaListener(topics = "${kafka.order.complete.topic.name}")

**public** **void** listenOrder(OrderEvent orderEvent) {

**if** (orderEvent.getActionName().equalsIgnoreCase("ORDER\_COMPLETE")) {

System.***out***.println("Your order has been successfully processed, you will receive in 2 days");

}

}

}

@Component

**public** **class** CancelOrderListener { 🡸 Cancel order

@KafkaListener(topics = "${kafka.order.cancel.topic.name}")

**public** **void** listenOrder(CancelEvent cancelEvent) {

**if**(cancelEvent.getActionName().equalsIgnoreCase("CANCEL\_ORDER")) {

System.***out***.println("Your order has been cancelled. Reason: "+cancelEvent.getReason());

}

}

}

@Component

**public** **class** CancelPaymentListener { 🡸 To cancel the payment or to reverse the transaction

@KafkaListener(topics = "${kafka.payment.cancel.topic.name}")

**public** **void** listenOrder(CancelEvent cancelEvent) {

**if**(cancelEvent.getActionName().equalsIgnoreCase("CANCEL\_ORDER")) {

System.***out***.println("Your order has been cancelled. Reason: "+cancelEvent.getReason());

}

}

}

@Component

**public** **class** CancelShipListener { 🡸 Cancel the shipping due to invalid address

@Autowired

**private** PaymentClient payClient;

@KafkaListener(topics = "${kafka.cancel.ship.topic.name}")

**public** **void** listen(CancelEvent cancelEvent) {

**if**(cancelEvent.getActionName().equals("CANCEL\_PAYMENT")) {

CancelRequest cancelReq = **new** CancelRequest();

cancelReq.setActionName("CANCEL\_PAYMENT");

cancelReq.setOrderId(cancelEvent.getOrderId());

cancelReq.setOrderName(cancelEvent.getOrderName());

cancelReq.setReason(cancelEvent.getReason());

payClient.reversePayment(cancelReq);

}

}

}

The Orchestrator Microservice also contains Feign clients to make rest calls to individual microservices.

@FeignClient(name = "InventoryServiceAPI", url = "${inventory.service.api.url}")

**public** **interface** InventoryRequestClient {

@PostMapping("/inventory/item")

**public** ResponseEntity<String> makeInventoryCall(@RequestBody ItemRequest itemRequest);

}

**Configuration – application.properties**

server.port=8085

spring.profiles.active=dev

# Kafka Consumer

spring.kafka.consumer.bootstrap-servers=localhost:9092

spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer

spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer

spring.kafka.consumer.group-id=saga-order-grp-id

spring.kafka.consumer.auto-offset-reset=latest

# The below line is important for Json Deserialization

spring.kafka.consumer.properties.spring.json.trusted.packages=\*

#Kafka Producer

spring.kafka.producer.bootstrap-servers=localhost:9092

spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer

spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer

# Open API Swagger documentation

springdoc.swagger-ui.path=/index.html

springdoc.swagger-ui.disable-swagger-default-url=true

#Application Specific

inventory.service.api.url=http://localhost:8082

payment.service.api.url=http://localhost:8083

shipping.service.api.url=http://localhost:8084

kafka.shipping.out.topic.name=saga-choreo-shipping-topic

kafka.order.complete.topic.name=saga-choreo-order-complete-topic

kafka.order.cancel.topic.name=saga-choreo-order-cancel-topic

kafka.payment.cancel.topic.name=saga-choreo-payment-cancel-topic

kafka.cancel.ship.topic.name=saga-cancel-ship-topic

kafka.inventory.out.topic.name=saga-inventory-out-topic

kafka.order.topic.name=saga-choreo-order-topic

**Advantages of Orchestration Pattern**

* Good for complex workflows involving many participants or new participants added over time.
* Suitable when there is control over every participant in the process, and control over the flow of activities.
* Doesn't introduce cyclical dependencies, because the orchestrator unilaterally depends on the saga participants.
* Saga participants don't need to know about commands for other participants. Clear separation of concerns simplifies business logic.

**Disadvantages of Orchestration Pattern**

* Additional design complexity requires an implementation of a coordination logic.
* There's an additional point of failure, because the orchestrator manages the complete workflow.