Temporal

# Partner Application Assessment

MAKE A COPY

## Purpose

This assessment challenges Partners to demonstrate the "ilities" required to ship a Temporal Application.
Criteria used to base Temporal maturity include:

Use of Temporal primitives
Completeness
Extensibility and maintainability to support subsequent versions
Test strategy
Cohesive Failure strategy
Integration considerations that impact scalability

## Constraints

Choose any Temporal SDK language

- External systems can be mocked or documented, but must describe a cohesive integration with the Temporal Application.
- Where requirements are not specific, creativity is encouraged.

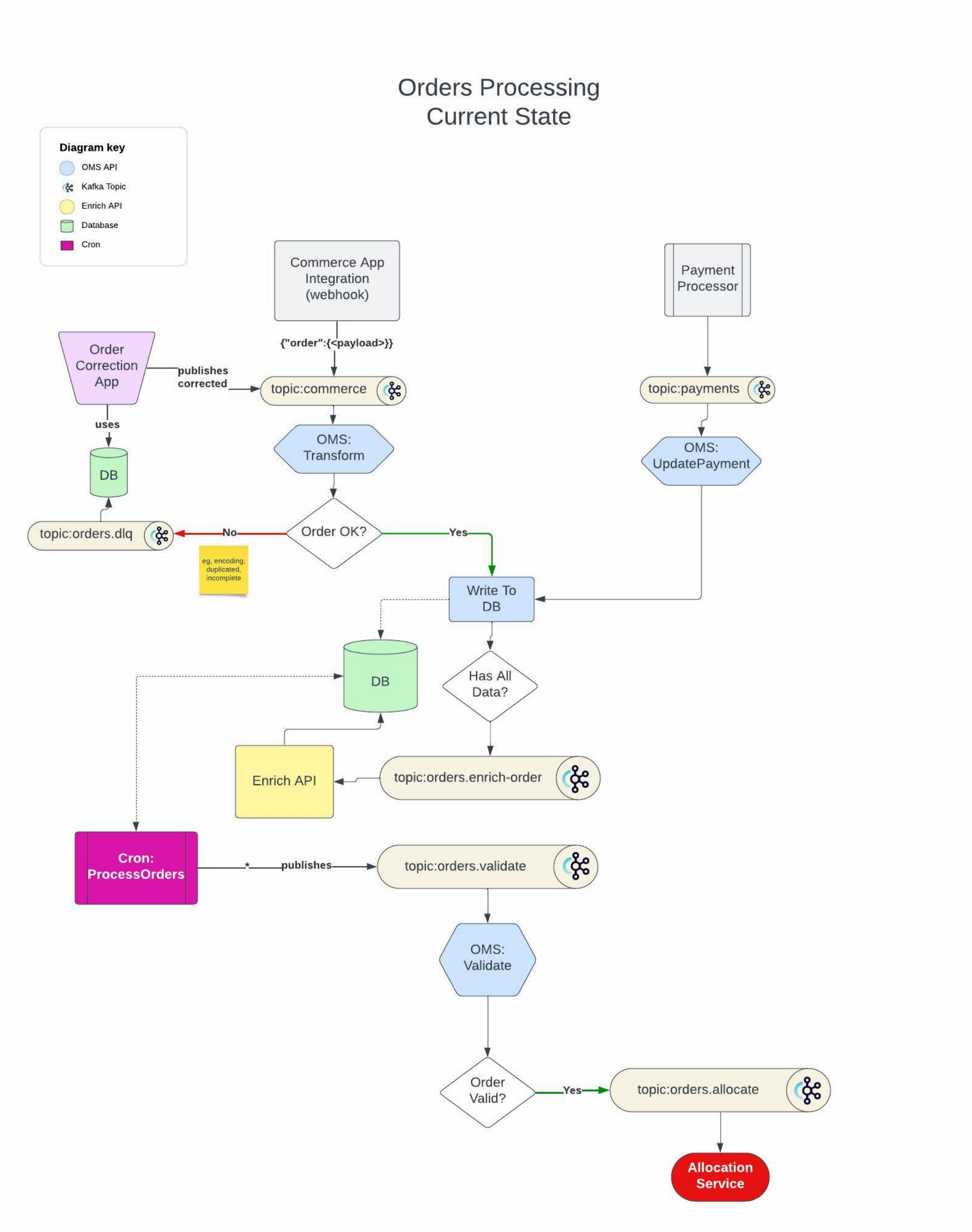
Temporal

# Order Processing Application [OMS]

Our current Clothing Order Management System aggregates data across various data sources via Kafka to set up orders for fulfillment.

Orders are completed by going through:

- Capture: Collect information
- Processing: Seeding the resources in all downstream services
- Fulfillment: Allocate Inventory and complete shipping



## Scope

We want to streamline just the Processing portion of this Order flow by improving how we capture and process the various inputs.

## Requirements

**Inputs** arrive from integration in different sequences. Sometimes, inputs might never arrive that would allow an Order to be processed.

- **TTL**: Orders should be forwarded to fulfillment within 30 days of submission.

    Expired orders should be written as such to the Customer dashboard storage.

**Order** validation: Invalid order data is sometimes received from our Commerce App service so we must try to validate that data against the API upon submission.

    Invalid orders must manually be corrected via our Support team so that the Order Processing can proceed.

- **Order enrichment**: Orders must be enriched with data from our internal systems to be fulfilled.

    See below for this enrichment.

**Payment Capture**: Happens between 1-30 days from authorization.

    Failure to receive a capture notification should cancel the order.

**Cancellation**: Orders are able to be cancelled up to the point of Payment capture.

  - If a payment has been captured, the Order will move on to fulfillment.

**Data Security**: Today we do not capture Personally Identifiable Information (PII) from our services, but we would like to begin collecting customer email addresses in future releases.

    We would like to understand how to protect this PII.

- **vNext [Risk collection]**: We need to support risk input collection and validation on our next release.

    Please describe in detail how we will make these changes after we go live with this initial release.

## Integration Details

We have integrations with other services that are required to process each Order, each carrying their own peculiar sets of requirements (rate-limiting, availability, etc).

- ***Commerce App*** is used as the store front.
    - This integration is set up today to receive webhook events from their service and push into our own systems.
    - Reads against this integration's API's are rate-limited at 150 RPS.
    - This API is used to validate and obtain richer order details we received after submission.
- ***Product Information Management System*** used to enrich order items
    - The SKU ID and Brand Code must be updated on each order item based upon its *Item ID*
    - The enriched Order details must be used for order *fulfillment*
- ***Payment Processing System*** is used to manage payments
    - The Retrieval Reference Number (RRN) is used to validate the payment against their API.
    - Their API has no rate-limits enforced.
    - Captured payments must be routed to the Order for processing completion.

Order details are written to a denormalized data storage to meet customer order dashboard UI requirements.

Temporal

## Data Specifications

### Inputs

*Commerce App webhook input*

```
 {
 "customer_id": <string>,
 "order": {
  "order_id": <string>,
  "items": [
   {
    "item_id": <string>,
    "quantity": <number>
   },
          ...
  ]
 }
}
```

*Payment Processor Service input*

```
 {
 "customer_id": <string>,
 "rrn": <string>,
 "amount_cents": <number>,
 "metadata": {
  "order_id": <string>
 }
}
```

### Output

The output for this system is a Kafka message dropped onto the order-fulfillment topic that carries along the *enriched* Order details, readying it for *allocation*.

*Order Fulfillment output message*

```
 {
 "customer_id": <string>,
 "order_id": <string>,
 "payment_details": {
  "rrn": <string>
 },
 "items": [
  {
   "item_id": <string>,
   "sku_id": <string>,
   "brand_code": <string>
  }, ...
 ]
}
```