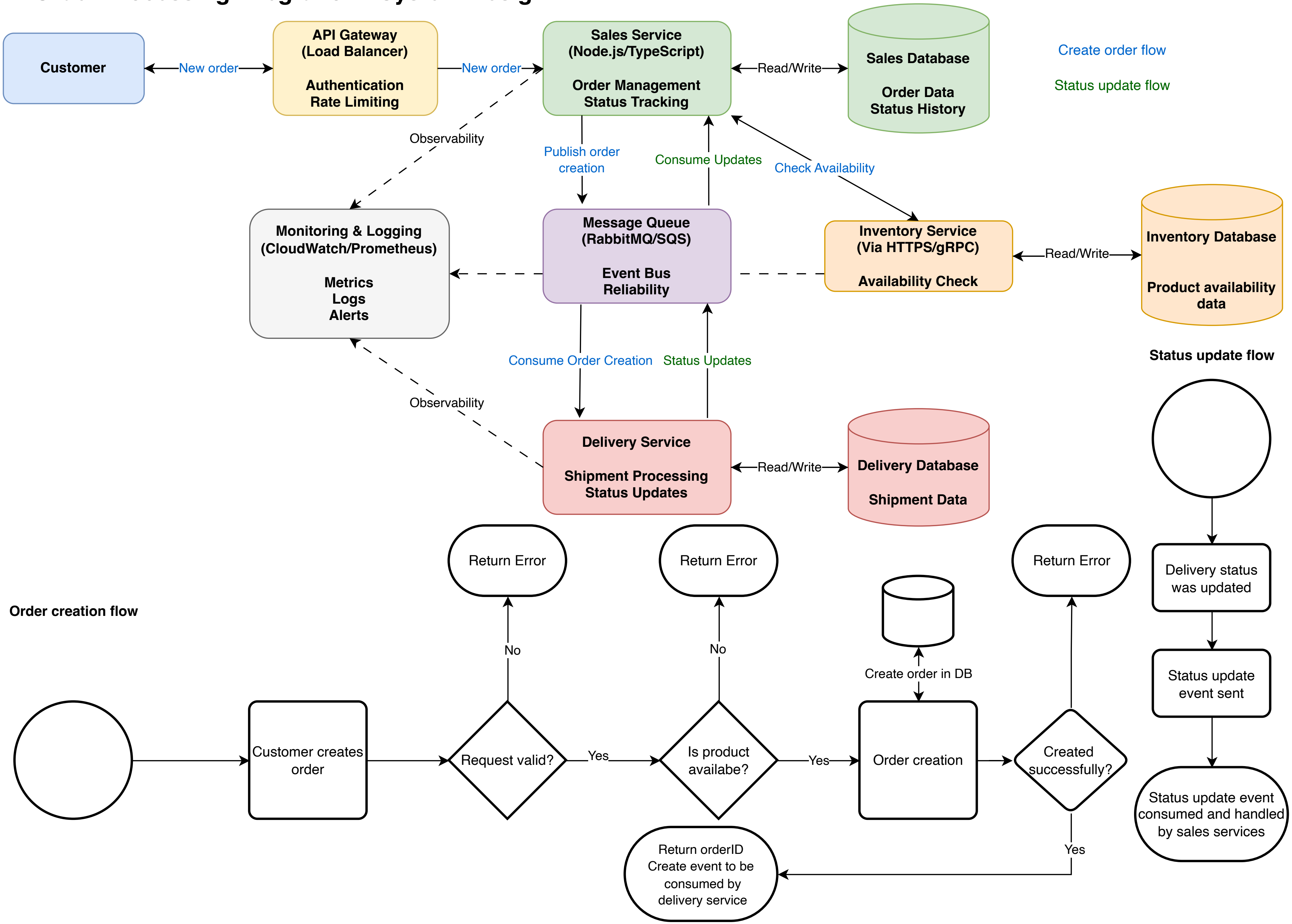
Order Processing Integration - System Design



Create order flow

Status update flow

Customer

New order

API Gateway
(Load Balancer)

Authentication
Rate Limiting

New order

Sales Service
(Node.js/TypeScript)

Order Management
Status Tracking

Read/Write

Sales Database

Order Data
Status History

Observability

Publish order creation

Consume Updates

Check Availability

Monitoring & Logging
(CloudWatch/Prometheus)

Metrics
Logs
Alerts

Message Queue
(RabbitMQ/SQS)

Event Bus
Reliability

Inventory Service
(Via HTTPS/gRPC)

Availability Check

Read/Write

Inventory Database

Product availability data

Consume Order Creation

Status Updates

Observability

Delivery Service

Shipment Processing
Status Updates

Read/Write

Delivery Database

Shipment Data

Order creation flow

Customer creates order

Request valid?

No

Return Error

Yes

Is product availabe?

No

Return Error

Yes

Order creation

Create order in DB

Created successfully?

Return Error

Yes

Return orderID
Create event to be consumed by delivery service

Status update flow

Delivery status was updated

Status update event sent

Status update event consumed and handled by sales services

System Components and Responsibilities

SERVICES COMPONENTS

Sales Service

- Receives and validates customer order requests
- Checks product availability via Inventory Service
- Creates orders in Sales Database
- Manages order status lifecycle (Pending Shipment → Shipped → Delivered)
- Publishes order creation events to Message Queue
- Returns order confirmation to customers with the order ID

Sales Database

- Stores order data (order ID, customer info, products, timestamps)
- Maintains order status history and state transitions
- Provides data persistence for order lifecycle management
- Supports queries for order retrieval and status updates

Delivery Service

- Consumes order creation events from Message Queue
- Initiates shipment processing workflow
- Creates shipment records in Delivery Database
- Manages shipment lifecycle and tracking
- Publishes status update events (Shipped, Delivered) to Message Queue

Delivery Database

- Stores shipment data and delivery information
- Tracks shipment status and logistics details
- Maintains delivery records and history
- Supports delivery service operations

Inventory Service

- Provides product availability checking functionality
- Validates if requested products are in stock
- Returns availability status to Sales Service
- Manages inventory data (external system)
- Supports synchronous API calls for real-time availability

Inventory Database

- Stores product availability data
- Maintains inventory levels and stock information
- Supports availability queries from Inventory Service

INFRASTRUCTURE COMPONENTS

API Gateway

- Acts as single entry point for external requests
- Handles authentication and authorization
- Implements rate limiting and request throttling
- Routes requests to appropriate backend services
- Provides SSL/TLS termination
- Distributes load across multiple service instances

Message Queue

- Provides asynchronous event-driven communication between services
- Decouples Sales and Delivery systems
- Ensures message delivery reliability and durability
- Supports event publishing (order creation, status updates)
- Implements message queuing with retry mechanisms
- Enables eventual consistency between systems
- Handles message routing and topic subscriptions

Monitoring & Logging

- Collects and aggregates application metrics (latency, throughput, error rates)
- Centralizes log collection from all services
- Provides real-time monitoring dashboards
- Generates alerts for system anomalies and failures
- Tracks system health and performance indicators
- Supports distributed tracing for request flows

Non-Functional Requirements

Scalability

Scalability is achieved by the ability to scale the number of instances running per service which in the sales service for example would be able to handle more incoming requests from customers or handle more status updates events

Security

To answer the security needs, we will have authentication on the API gateway to ensure all requests from customers through the APIare authenticated. Authentication will also be applied to the message queue service as well for cross service interactions. Rotating keys can be used with a secret manager to obtain the credentials to use the services. Regarding the traffic adding an SSL encryption layer will prevent data from being read. For instance making sure all requests are done through HTTPS.

Reliability

Using a message queue with persistency and ack abilities to manage events between services will make sure all messages are handled and no data is lost. Adding some policy for a dead letter queue and the ability to digest the messages later will add another layer of handling data loss/issues. For blocking operations between services the usage of HTTPS or gRPC would give us a request-response model that we can decide on the same request if to reject or approve the order. Note: I am assuming that the inventory service is a crucial that service that is always up and orders can't be created without it as it is part of the requirement for an order to be created.

Logs & Metrics

We can use the different available logging and monitoring solution to emit metrics of the different running process and containers. Other than that emitting logs tracing different steps of the order lifecycle with identifiers would help analyze and trace the flow.

Message duplicates handling

We can handle duplicate messages on different levels. One would be writing our event handling in such a way that would check if the requested change/update was already done. For example if this is a status update that has the same timestamp and same value of an update that is already stored the message will be ignored. Or if an order already has a delivery created it won't be created again. Or that receiving the change multiple times won't matter. Another way would be to have a shared cache between consumers, allocate an ID once sending the message and checking in cache before handling the event. Last way there are plugins that can be added that can prevent deduplication in queues at the exchange or queue itself.