Deakin University

Smart Warehouse Inventory and Delivery Management System

Project Proposal

Student Name: Ocean Ocean

Student ID: S223503101

Date of Submission: 19 July,2025

Document Version 1.0

High-Level Problem / Problem Description

- Modern warehouses often struggle with real-time stock monitoring, delivery scheduling, and maintaining optimal inventory levels. Manual systems or semi-automated solutions lack scalability and cannot adapt efficiently to demand spikes, such as during peak sales periods.
- This project proposes the development of a Smart Warehouse Inventory and Delivery Management System that utilizes IoT sensors, event-driven microservices, and cloud scalability. The goal is to automate stock monitoring, streamline order processing, and enable scalable delivery scheduling.
- Unlike traditional inventory systems, this solution uses simulated sensor data, Node-RED for event-based flow control, and scalable microservices deployed on AWS. The design supports autoscaling of services like order processing and delivery notifications, ensuring consistent performance during high demand.

Solution overview

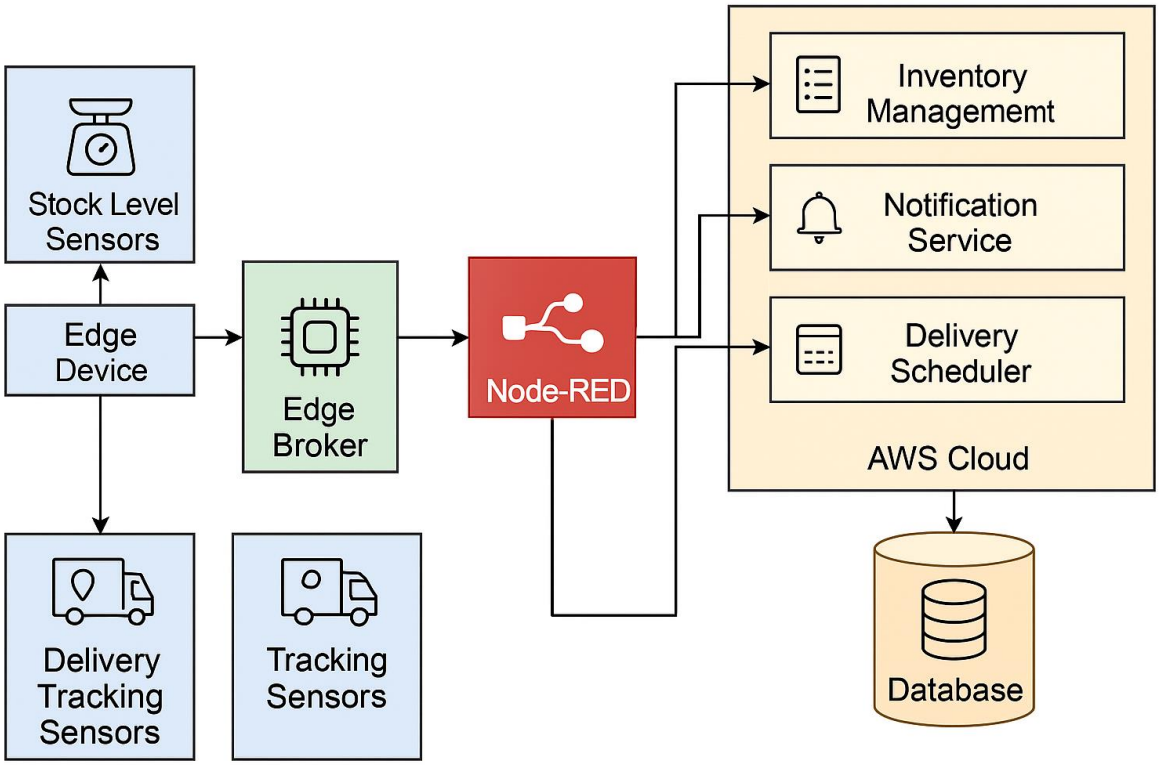
The proposed system will:

- Monitor inventory levels using simulated IoT sensors.
- Trigger automatic reordering when stock falls below threshold.
- Schedule deliveries via a scalable event-based microservice architecture.
- Store data in cloud databases (e.g., AWS DynamoDB).

The architecture includes sensors (simulated with Node.js), a data processing pipeline using Node-RED, microservices in Node.js, and deployment on AWS with automatic scaling.

High-level block diagram:

- Sensor Nodes (Simulated stock readers)
- Node.js Data Collector
- Node-RED Flow Processor
- Microservices (Inventory, Order, Delivery)
- AWS Infrastructure (EC2, Lambda, S3, DynamoDB)



Aggregation/filtering includes averaging stock levels, filtering noise, and applying reorder thresholds. The system will be scalable through AWS EC2 Auto Scaling Groups and Lambda triggers.

Testing Plan:

- Unit testing for microservices
- Load testing on AWS with simulated traffic
- Functional testing for event handling in Node-RED
- Security testing using IAM and HTTPS endpoints

Implementation Plan

Hardware/Simulation:

- Simulated weight or RFID sensors via Node.js scripts
- MQTT or HTTP for communication
- Node-RED for rule-based flows

Communication:

- MQTT for sensor-to-gateway messaging
- HTTP/REST for microservices

Data Design:

- Inventory Data: item_id, stock_level, last_updated

- Order Data: order_id, item_id, quantity, status
- Delivery Data: delivery_id, address, status

Storage:

- DynamoDB for scalable, document-based storage
- S3 for logs and backups

Cloud Deployment:

- EC2 for core services
- AWS Lambda for auto-triggered scaling
- API Gateway for microservice exposure
- IAM roles for secure access control

Project Plan

Week	***Planned Activities***
Week 1	*Finalize project topic (Smart Warehouse), set up GitHub repo, begin drafting project plan*
Week 2	*Design system architecture and high-level block diagram, research AWS & Node-RED integration*
Week 3	*Develop simulated sensor scripts using Node.js to represent stock levels*
Week 4	*Implement Node-RED flows to process sensor data and trigger events*
Week 5	*Create core microservices: Inventory Manager, Order Handler, Delivery Scheduler*
Week 6	*Deploy microservices to AWS EC2, configure AWS Lambda and Auto Scaling Groups*
Week 7	*Set up DynamoDB (for inventory & orders), S3 for log storage, secure access with IAM*
Week 8	*Conduct functional, integration, and load testing of the full system*
Week 9	*Finalize documentation, gather screenshots/logs, export plan to PDF, submit to OnTrack*

Potential Risks:

- AWS billing limits or misconfiguration
- Delays in integrating services
- Inconsistent simulation data

Cloud deployment issues due to IAM roles or policy errors