📄 Technical Solution Document

Project Name: Real-Time Event-Driven Data Pipeline for E-Commerce Analytics
Author: Marzuk Sanni Entsie
License: MIT
Last Updated: 11th April 2025

---

1. Introduction

1.1 Purpose

This document describes the architecture, components, and operational flow of a real-time, event-driven data pipeline developed for an e-commerce platform. The solution is designed to process continuously arriving order-related data in Amazon S3, compute business KPIs, and store them in Amazon DynamoDB for real-time analytics.

1.2 Scope

This pipeline enables:

- Real-time ingestion and validation of flat files from S3
- Transformation and KPI calculation
- Storing results in optimized DynamoDB tables
- Automated execution using AWS Step Functions
- Logging, notifications, and error handling

1.3 Definitions, Acronyms, and Abbreviations

Term	Description
ECS	Elastic Container Service
S3	Simple Storage Service
KPI	Key Performance Indicator
Step Functions	AWS orchestration service

Term	Description
SNS	Simple Notification Service
ETL	Extract, Transform, Load

1.4 References

- AWS Documentation
- Project repository README

1.5 Overview

The system processes three primary data types: products, orders, and order items. It validates, transforms, and computes business KPIs at both category and order levels.

---

2. Architectural Representation

2.1 Description of Architecture

This solution follows a microservices-based, event-driven design. It uses ECS containers for modular processing and Step Functions for workflow orchestration. Each service is responsible for a dedicated task in the data pipeline.

---

3. Architectural Goals and Constraints

3.1 Goals

- Modular and scalable architecture
- Near real-time KPI computation
- Fault-tolerant and observable system
- Automated workflow with retries and failure handling

3.2 Constraints

- Must use AWS-native services
- Data format must be flat files (CSV)

- Must be containerized using ECS
- Must store KPIs in DynamoDB

---

4. Use-Case View

4.1 Use-Case Model

Primary Use-Case: Process Incoming Data File

Actors:

- S3 (data source)
- ECS Containers (processors)
- Step Functions (orchestrator)
- DynamoDB (storage)
- SNS (notifications)

4.2 Use-Case Realizations

Each ECS container runs specific logic:

- Validation Container: Ensures data integrity and schema conformity.
- Transformation Container: Computes KPIs and formats output.
- Computation Container: (Optional for extended metrics or post-processing)

---

5. Logical View

5.1 Logical Components

- validate-service: Checks for missing fields, data types, referential links
- transform-service: Computes KPIs per specifications
- compute-service: Further computations (optional/custom KPIs)
- lambda: Trigger Step Functions from S3 PUT events

---

6. Process View

6.1 Concurrent Processes

- Step Functions handles orchestration with conditional branching and parallel processing
- CloudWatch captures logs concurrently from ECS tasks

6.2 Step Function Explanation

The Step Function executes the workflow in the following sequence:

1. Parallel Validation Step:
    - Incoming files (e.g., products, orders, order_items) are validated in parallel using ECS containers.
    - If any validation fails, the workflow is immediately terminated.
    - An error notification is sent via SNS.
2. Transformation Step:
    - Upon successful validation, the transformation container computes both Category-Level and Order-Level KPIs.
    - Results are written to two separate DynamoDB tables.
3. Success Notification:
    - Once KPIs are stored successfully, an SNS notification email is sent confirming completion.

---

7. Deployment View

7.1 Component Mapping

Component	AWS Service
Lambda Trigger	AWS Lambda

Component	AWS Service
Validation, Transformation	Amazon ECS
File Storage	Amazon S3
Metrics Storage	Amazon DynamoDB
Monitoring	Amazon CloudWatch
Workflow Orchestration	AWS Step Functions
Notification System	Amazon SNS

---

8. Implementation View

8.1 Development Environment

- Python 3.x
- Docker
- AWS CLI
- VSCode or PyCharm

8.2 Configuration Management

- Code stored in GitHub
- Deployment via shell script (scripts/push to cloud.sh)
- ECS task definitions managed in scripts/task-definitions

---

9. Data View

9.1 Schema

Refer to assets/images/ERD.png – *Figure 2: Entity Relationship Diagram*

9.2 DynamoDB Table Design

Category-Level KPI Table

Field	Type	Key
category	String	Partition key
order_date	Date	Sort key

Order-Level KPI Table

Field	Type	Key
order_date	Date	Partition key

Secondary indexes may be added to optimize queries based on read patterns.

---

10. Setup Instructions

To set up and deploy this pipeline:

1. Build Docker Images
2. Push Docker Images to Amazon ECR
3. Pull Images into ECS & Register Task Definitions
    - Add definitions to ECS via AWS Console or CLI using files in scripts/task-definitions.
4. Set Up IAM Roles and Policies
    - Create execution roles with access to S3, ECS, DynamoDB, SNS, and CloudWatch.
5. Create VPC, Subnets, and Security Groups
    - For ECS task networking and security.
6. Provision DynamoDB Tables
    - Create the two KPI tables with proper partition/sort keys as defined.
7. Deploy Step Function
    - Use the scripts/step_function.json to define and deploy the state machine.

8. Configure Lambda Trigger
    - A Lambda function should trigger the Step Function upon file upload to S3.

---

11. Quality

11.1 Attributes

Quality Attribute	Implementation Description
Usability	Single script deployment, CLI driven
Reliability	Built-in retries, failure paths, and alerts
Maintainability	Modular ECS services and clearly separated concerns
Scalability	Serverless Lambda trigger + container orchestration
Security	IAM role-based access, private subnets
Observability	CloudWatch metrics, logs, SNS alerts for success/failure

---

12. Appendices

A. Glossary

- KPI: Key Performance Indicator
- ECS: Elastic Container Service
- Step Functions: AWS service to orchestrate workflows

B. Revision History

Version	Date	Description
1.0	11th April 2025	Initial draft