OutOfTheBoxEA

chaala_easy_bro

Out of the Box
ACADEMY

# ACADEMY REALTIME SCENARIOS

Presentation by OOB Academy

www.outoftheboxacademy.com

*Join our SAP CPI course to Learn How to build these scenarios by an Industry Expert*

# Realtime Scenario 1 - HR Timesheets Integration

## Real-Time Business Scenario – Employee Timesheet Submission

**Scenario:**

Employees of an organization use a **custom-built web application** to log their daily work hours. This data must be sent in real-time to the **HR department's system**, which processes valid timesheets and stores them in a shared folder in **CSV format**. Invalid submissions (e.g., hours worked > 12) must be **logged separately** for audit and compliance.

| Business Scenario | Integration Design | Business Benefit |
|---|---|---|
| Employees log timesheets via web. Data must be validated before being sent to HR system. Valid entries go to a shared folder (CSV), invalid ones are saved in a separate error folder. | - **Source System**: Employee Web App<br>- **Target System**: On-Premise HR System (CSV files)<br>- **Adapters**:<br>• HTTP **Sender Adapter** – Receives JSON data<br>• File **Receiver Adapter** – Writes CSV files to shared folder<br>- **Data Structures**:<br>• Input JSON: `{"employeeId":"E123","date":"2025-07-16","hours":9}`<br>• Intermediate XML: `<Timesheet><EmpID>E123</EmpID><Date>2025-07-16</Date><Hours>9</Hours></Timesheet>`<br>• Final CSV: `E123,2025-07-16,9`<br>- **Palettes Used**:<br>• **JSON to XML Converter**: convert input to structured format<br>• **Filter**: check if hours ≤ 12<br>• **Router**: direct valid to HR path, invalid to error folder<br>• **XML to CSV Converter**: for HR import format | - Enables **real-time** processing of timesheets<br>- Automates **HR data ingestion**, removing manual steps<br>- Enforces **business validation rules** (e.g., work hours ≤ 12)<br>- Improves **data accuracy** and **audit compliance**<br>- **Segregates** invalid records to simplify error handling<br>- Provides **scalability** for future enhancements like payroll integration |

# Real-Time Scenario 2 – Product Shipment Orders Processing

## 📘 Real-Time Scenario – Product Shipment Orders Processing

| Business Scenario | Integration Design | Business Benefit |
| --- | --- | --- |
| Logistics team receives a daily shipment file containing multiple product orders. Each order needs to be validated (filter), routed based on region, converted to a standard format, and saved as individual XML files. | - **Source System:** Logistics shared folder<br>- **Target System:** Regional warehouse folders<br>- **Adapters:**<br>• File **Sender Adapter** – Picks up batch shipment XML<br>• File **Receiver Adapter** – Delivers XML to region folders<br>- **Palettes Used:**<br>• **Splitter:** XML Splitter using XPath `/Shipments/Order`<br>• **Filter:** Check if quantity > 0<br>• **Router:** Route to EU or US warehouse based on `<Region>` tag<br>• **XML to JSON Converter** (if required by downstream)<br>- **Data Structures:**<br>• Input XML: `<Shipments><Order><ID>123</ID><Region>EU</Region><Qty>10</Qty></Order>...</Shipments>`<br>• Output (per file): `<Order><ID>123</ID><Region>EU</Region><Qty>10</Qty></Order>` or converted JSON | - Automatically processes bulk shipment files<br>- Ensures only valid orders (Qty > 0) are processed<br>- Intelligent routing improves delivery efficiency<br>- Data conversions align with regional systems<br>- Reduces manual errors and accelerates warehouse fulfillment |

# Real-Time Scenario 3 – Customer Feedback Archival

## Real-Time Scenario – Customer Feedback Archival

| Business Scenario | Integration Design | Business Benefit |
|---|---|---|
| Customer feedbacks are collected daily in a single XML file from a web survey tool. Each feedback entry must be validated (non-empty comment), enriched with metadata (company name, received date), converted to JSON, and stored as separate files in a shared folder for analysis. | - **Source System**: Survey system (XML batch file)<br>- **Target System**: Data analytics archive (JSON files)<br>- **Adapters**:<br>• File **Sender Adapter** – picks up the feedback XML file<br>• File **Receiver Adapter** – stores converted JSON files<br>- **Palettes Used**:<br>• **XML Splitter**: XPath `/Feedbacks/Entry`<br>• **Filter**: `comment != null` or `comment != ""`<br>• **Content Modifier**: adds `CompanyName = "XYZ Ltd"`, `ProcessedDate = ${date:now}`<br>• **XML to JSON Converter**: transforms each valid XML to JSON | - Automates feedback processing<br>- Filters out incomplete or blank submissions<br>- Adds context data for business insight<br>- Converts to JSON for easier integration with BI tools<br>- Enables faster, structured customer sentiment analysis |

# Real-Time Scenario 4 – Sales Order Processing with Conditional Enrichment and Dynamic Routing

## Real-Time Scenario – Sales Order Processing with Conditional Enrichment and Dynamic Routing

| Business Scenario | Integration Design | Business Benefit |
| --- | --- | --- |
| A central e-commerce system sends sales orders to SAP CPI. Each order must be enriched with multiple headers and properties like region-specific tax codes, dynamic pricing tier, partner ID (based on customer segment), and must also carry dynamically generated tracking URLs and unique file names. These enriched orders are then routed to the appropriate regional SAP S/4HANA systems. | - **Source**: E-commerce API (JSON via HTTP)<br>- **Target**: SAP S/4HANA systems (multiple)<br>- **Adapters**:<br>• HTTP **Sender Adapter**<br>• SOAP or IDoc **Receiver Adapters** per region<br>- **Palettes Used**:<br>• **Content Modifier** (Multiple Usage):<br>• Set dynamic **message headers**: `Region`, `OrderTier`, `TrackingURL`, `CorrelationID`<br>• Set **message properties**: `PartnerID`, `TaxCode`, `CountryCode`, `ProcessedDate`<br>• Use expressions: `${xpath}`, `${date:now}`, `${header.someValue}`, `${property.someProp}`<br>• Optional body overwrite with templated payload<br>• **Router**: route based on `Region` header<br>• **JSON to XML Converter**: before sending to backend | - Automates dynamic enrichment with no manual config<br>- Adjusts for region-specific compliance (tax codes, partner IDs)<br>- Builds smart routing using message metadata<br>- Enables detailed tracking and logging per message<br>- Scales across multiple backend systems with one reusable flow |

# 🟦 Real-Time Scenario 5 – Regional Sales Order File Creation

### 🟦 Business Requirement – Regional Sales Order File Creation

#### 🏬 Use Case Overview

A global e-commerce company operates across multiple regions: **US, EU, and APAC**. Sales orders are placed by customers in these regions and must be integrated with a **regional File Server** for downstream **order creation**. Each region has its **own standards** for:

- **Product Codes**
- **Currency Codes**
- **Tax Rates**

These attributes must be **mapped and enriched correctly** before generating the final files.

---

#### 🔁 Processing Requirements

To ensure accurate data transformation and system stability, the following integration logic must be followed:

- ✅ **ODATA Lookup**: Use ODATA API `/sap/opu/odata/sap/API_PRODUCT_SRV/` to fetch product details (e.g., regional codes, tax classifications, etc.)
- ✅ **Pagination and Looping**: Handle API pagination using **Looping Process Call** to fetch all relevant product data.
- ✅ **Product Grouping**: Each output file must contain **a maximum of 20 product records** to prevent file size overflow and adhere to the File Server limits.
- ✅ **Final Output**: Enriched product data must be **written to the file system**, region-wise, using **File Receiver Adapter** in respective regional folders.

# 🟦 Real-Time Scenario 6 – Supplier Creation via SAP CPI (Integration Testing Use Case)

### 🟦 Business Requirement – Supplier Creation via SAP CPI (Integration Testing Use Case)

#### 🏬 Use Case Overview

A company aims to **create new supplier records in SAP S/4HANA** using data from an external application. Before rolling out the full integration, the development team wants to **test and validate** the end-to-end flow using **Postman** as the request sender.

This testing setup will help confirm:

- API accessibility
- Data structure validation
- SAP S/4HANA integration via SAP CPI
- Error handling and monitoring

---

#### 🔁 Processing Requirements

To enable smooth and reliable testing of the supplier creation interface:

- ✅ **Test Trigger**: Postman sends a **sample HTTP request** to SAP CPI, containing supplier details.
- ✅ **Target API**: SAP CPI sends the request to SAP S/4HANA using ODATA service `/sap/opu/odata/sap/API_SUPPLIER_SRV/` via **HTTP Receiver Adapter**.
- ✅ **Error Handling**: If the iFlow fails (e.g., due to mapping or authorization errors), an **exception subprocess** will catch the failure and **send an email alert via Gmail** to the development team.
- ✅ **Response Propagation**: The HTTP response from SAP S/4HANA (success or error) will be captured and returned to Postman for real-time validation.

# 🟦 Real-Time Scenario 7 – Supplier Data Retry & Persistence using Data Store

## 🟦 Business Requirement – Supplier Data Retry & Persistence using Data Store

### 🏬 Use Case Overview

The company's supplier onboarding process requires sending supplier records from a third-party application to **SAP S/4HANA** using the ODATA API: `/sap/opu/odata/sap/API_SUPPLIER_SRV/`. However, due to frequent **intermittent network/API failures**, some supplier creation requests fail and are lost if not reprocessed.

To address this, the integration must be enhanced with **Data Store logic** to persist failed payloads and support retries.

---

### 🔁 Processing Requirements

To ensure reliable data integration and minimize data loss:

- ✅ **Write to Data Store on Failure**:
  When a supplier creation request fails due to S/4HANA unavailability or API timeout, the payload should be written to the **SAP CPI Data Store** using a unique key (`SupplierID` + timestamp).
- ✅ **Retry Mechanism via Read Operation**:
  A scheduled iFlow will **read unprocessed supplier records from the Data Store**, reattempt the API call to S/4HANA, and update the record status (success or permanent failure).
- ✅ **Success Criteria**:
  If the retry is successful, the entry should be **deleted from the Data Store**. If it fails again, the system should **send an alert (email)** for manual investigation.

# 🟦 Real-Time Scenario 8 – Job Requisition Archival with DB Logging via JDBC

### 🟦 Business Requirement – Job Requisition Archival with DB Logging via JDBC

#### 🏬 Use Case Overview

A multinational organization uses **SAP SuccessFactors** for managing job requisitions. Every time a new **Job Requisition** is created in SuccessFactors, the record needs to be integrated with a downstream system and simultaneously **logged in a local SQL database** for auditing and compliance.

This integration must ensure:

- The job requisition data is reliably archived.
- Any failure in downstream posting is captured and logged.
- An alert is sent to HR admins when failures occur.

---

#### 🔁 Processing Requirements

- ✅ **Data Source**: SAP SuccessFactors Job Requisition API (`/odata/v2/JobRequisition`) is queried on a schedule to fetch **new job requisitions**.
- ✅ **DB Archival via JDBC**: For each new record, key job requisition data (Job ID, Title, Location, Department, etc.) is **inserted into a PostgreSQL/SQL database** via **JDBC Adapter**.
- ✅ **Downstream Posting (Optional)**: Data can optionally be forwarded to another system via SOAP/HTTP after DB insert.
- ✅ **Exception Handling**:
  - If the **DB insert fails**, the payload and error are logged to a **Data Store** or stored as error file.
  - A **Gmail adapter** sends an alert email with the error and Job ID to the HR IT team.

# Real-Time Scenario 9 – Product Master Sync from SQL DB to AMQP Queue

### Business Requirement – Product Master Sync from SQL DB to AMQP Queue

**Use Case Overview**

A retail enterprise maintains its **product master data** in a centralized **on-premise SQL database**. This data needs to be **synchronized periodically** with multiple downstream systems such as mobile apps, POS systems, and partner channels.

To enable real-time propagation and decoupled distribution, the product records are **read from the SQL DB using JDBC**, and each product record is then **published as a message to an AMQP-based message broker** (e.g., RabbitMQ or Azure Service Bus).

---

**Processing Requirements**

- ✅ **JDBC Read Operation**:
  SAP CPI should use **JDBC Adapter (Select)** to query new or updated product records from the **Product_Master** table.
- ✅ **Loop and Publish**:
  Each product record is **processed individually** (via Splitter or Looping Process Call) and sent to the **AMQP Adapter**, which publishes the message to a configured **exchange/queue**.
- ✅ **Message Structure**:
  Product payload should be structured in **JSON format** before publishing to meet downstream consumer needs.
- ✅ **Error Handling**:
  If AMQP delivery fails:
  - The message should be logged to **Data Store** for retry.
  - A **notification email** should be triggered via the **Gmail Adapter** with the Product ID and error details.

www.outoftheboxacademy.com

# Real-Time Scenario 10 – Secure Customer Registration via AMQP with DB Enrichment and Logging

### 📘 Business Requirement – Secure Customer Registration via AMQP with DB Enrichment and Logging

#### 🏬 Use Case Overview

A digital services provider receives **new customer registration events** via an **AMQP-based message queue** from its website and mobile apps. Each message contains encrypted customer data (e.g., name, email, region code). To complete the registration process:

1. The encrypted payload must be **decrypted** in SAP CPI.
2. The **region code** must be **enriched** from a local SQL database to fetch the full region name.
3. The final **enriched and validated customer record** must be **stored in a customer audit table** for analytics and compliance.

---

#### 🔁 Processing Requirements

- ✅ **Read from AMQP Queue**:
  SAP CPI uses the **AMQP Adapter (Sender)** to receive encrypted JSON messages from the queue.
- ✅ **Data Decryption**:
  Use a **Groovy script** with AES/GPG/RSA logic to **decrypt sensitive fields** like customer name and email.
- ✅ **DB Lookup (JDBC Read)**:
  Based on the region code (e.g., `EU`, `US`, `APAC`), a lookup is performed against a **Region_Master** SQL table using JDBC.
- ✅ **DB Write (JDBC Insert)**:
  Final enriched customer data (with decrypted fields and full region name) is inserted into a **Customer_Registration_Log** table using **JDBC Write**.
- ✅ **Exception Handling**:
  - Any decryption or DB failure is caught in an **Exception Subprocess**.
  - The failed message is logged to **Data Stor**
  - An **email notification** is sent using the **Gmail Adapter** with customer ID and error trace.

www.outoftheboxacademy.com