Integration Requirements

Group Centre of Excellence Blockchain

Integration Guide
Vendor Order Module 1.0

Reference: **Vendor Order Module 1.0**
Date Issued : **Q3 2023**
Distribution : **DHL ITS**
Document version : **1.0**
Status : **Draft**

Table of Contents

Contents

Introduction

This guide has been crafted to serve as a comprehensive resource for developers and system administrators seeking to smoothly integrate the Vendor Order Application into their current systems and operational processes. Whether your goal is to streamline data processing, automate routine tasks, or enhance data compatibility, this document will provide you with a step-by-step roadmap to effectively integrate and harness the functionalities offered by the Vendor Order Application.

About the Vendor Order Application

The Vendor Order Application plays a pivotal role in aggregating shipment details from IBM, utilizing both Kafka and API channels, in two primary formats: JSON and PDF. Subsequently, the application proceeds to transform this data before forwarding it to the Bless system. To ensure the efficiency of this complex process, microservices are employed, orchestrating various components to seamlessly handle the entire operation.

Intended Audience

This integration guide has been specifically crafted for developers, system architects, and technical experts who bear the responsibility of seamlessly integrating the Vendor Order Application into their existing software systems, data pipelines, or operational workflows. Whether you're a software engineer tasked with constructing a custom integration or a system administrator overseeing data processes, this guide furnishes the essential information required to achieve a successful integration of the Vendor Order Application.

Key Benefits

a) Automation: Facilitate automated data transformation workflows, reducing manual interventions and saving time.
b) Interoperability: Act as a bridge between disparate systems, facilitating seamless data exchange and collaborative efforts.
c) Enhanced Productivity: Empower your team by automating mundane data transformation tasks, allowing them to focus on value-added activities.
d) Data Enrichment: Augment your data with supplementary information from external sources, elevating its value and precision.
e) Reliability: We've implemented a retry mechanism to enhance stability and ensure data processing reliability.

Prerequisites

Before proceeding with the integration, please ensure that you have the following prerequisites in place:

- Access to the Vendor Order Application: Obtain the necessary access credentials or licensing information to use the Vendor Order Application.
- Basic Technical Proficiency: Familiarity with data formats, APIs, and integration concepts will be beneficial.
- Understanding of the Systems: A clear understanding of the existing systems, data sources, and target formats will facilitate the integration process.

Integration Touchpoints in Scope

- Setup MySQL Database
- Setup Docker Desktop
- Setup & Deploy Python Kafka Client
- Setup & Deploy Vendor Order Module
- Setup & Deploy Web Vendor Order Module

Integration Steps

Setup MySQL Database

1. ### Install MySQL:

   Visit the MySQL Downloads page.
   Download the MySQL Community Server based on your operating system (e.g., Windows, macOS, or Linux). Choose the appropriate version (usually the latest stable version is recommended).
   Follow the installation instructions for your OS to install MySQL.

2. ### Create Database:

   Create your database named DPDHL_DB using following command:

```
CREATE DATABASE DPDHL_DB;
```

3. ### Create Table:

   Create your table using following schema:

```
CREATE TABLE dt_ibm_shipment (
    Id INT AUTO_INCREMENT PRIMARY KEY,
    shipment_id VARCHAR(50),
    createddate DATETIME DEFAULT CURRENT_TIMESTAMP,
    COMINV_status BIT DEFAULT 0,
    COMINV_proces_date DATETIME,
    COMINV_retry_count INT DEFAULT 0,
    shipment_traking_no VARCHAR(255),
    invoice_number VARCHAR(255),
    file_names VARCHAR(255),
    COMINV_DOC_status BIT DEFAULT 0,
    COMINV_DOC_proces_date DATETIME,
    COMINV_DOC_retry_count INT DEFAULT 0
);
ALTER TABLE dt_ibm_shipment
ADD CONSTRAINT shipment_id_unique UNIQUE (shipment_id);
```

Setup Docker Desktop

1. ### Install Docker:

   Visit the Docker website and download the appropriate Docker version for your operating system (Windows, macOS, or Linux). Choose the stable version for production use.

Follow the installation instructions for your OS. On Windows and macOS, Docker typically comes with Docker Desktop, which includes the Docker Engine and other tools.

2. Verify Docker Installation:

After installation, verify that Docker is installed correctly by opening your terminal or command prompt and running the following command:

```
docker –version
```

You should see the Docker version displayed in the output.

3. Start the Docker Daemon:
On most systems, Docker should start automatically after installation. If it doesn't, you may need to start the Docker daemon manually:

On Linux, you can start Docker with:

```
sudo systemctl start docker
```

On Windows and macOS, Docker should start when you launch Docker Desktop.

Setup & Deploy Python Kafka Client

1. Create Docker Images:
Go to the project root folder. Create a docker image of this application using following command:

```
docker build -t <username>/python-kafka-client:1.2 .
```

Here <Username> will be the user name of docker hub if you want to push this image to docker hub. Otherwise, no need to provide username if you want to run this image to your local machine.

2. Push Image:

To push this image to docker hub, you have to use following command:

```
docker push yeasinmahi/python-kafka-client:1.2
```

3. Pull Image:

To pull image to the production/beta server please use following command to pull this image.

```
docker pull yeasinmahi/python-kafka-client:1.2
```

Note: You have to have the docker install where you want to deploy the docker applications.

4. Run the application:

To start application, you need a docker compose file. So, you need to create a file named docker-compose.yml and past the following code to that file.

```
version: "3.1"
services:
  python-kafka-client:
    image: "yeasinmahi/python-kafka-client:1.2"
    container_name: "python-kafka-client"
    restart: always
    environment:
        bootstrap.servers: broker-4-xls69gx4n2mn4wmv.kafka.svc11.us-
south.eventstreams.cloud.ibm.com:9093,broker-2-
xls69gx4n2mn4wmv.kafka.svc11.us-
south.eventstreams.cloud.ibm.com:9093,broker-0-
xls69gx4n2mn4wmv.kafka.svc11.us-
south.eventstreams.cloud.ibm.com:9093,broker-3-
xls69gx4n2mn4wmv.kafka.svc11.us-
south.eventstreams.cloud.ibm.com:9093,broker-1-
xls69gx4n2mn4wmv.kafka.svc11.us-
south.eventstreams.cloud.ibm.com:9093,broker-5-
xls69gx4n2mn4wmv.kafka.svc11.us-
south.eventstreams.cloud.ibm.com:9093
        topic: s2d-spo-from-ibm-to-supplier2

        ## Consumer specific properties
        group.id: ibmeventstreams_starter_app_group
        auto.offset.reset: earliest
        enable.auto.commit: 'false'
```

```
    consumer.timeout: 10
    key.deserializer:
org.apache.kafka.common.serialization.StringDeserializer
    value.deserializer:
org.apache.kafka.common.serialization.StringDeserializer

    ## Producer specific properties
    acks: all
    key.serializer:
org.apache.kafka.common.serialization.StringSerializer
    value.serializer:
org.apache.kafka.common.serialization.StringSerializer

    ## Optional security options
    security.protocol: SASL_SSL
    ssl.protocol: TLSv1.2
    sasl.jaas.config:
org.apache.kafka.common.security.plain.PlainLoginModule required
username = "token" password =
"p8PGJ250PRGTUicpOReYoitcjsCLFS8TAwcUnlUAxXHl"
    sasl.username: token
    sasl.password: AKDbiBbl-WgTluX5e8hNRPf7-
bhmE02RRNYBcJGdpoLX
    sasl.mechanism: PLAIN
    ssl.enabled.protocols: TLSv1.2
    ssl.endpoint.identification.algorithm: HTTPS

    # exception request
    exception.host: https://uat-transcomm.api.blesstrace.com

    ## Log Specific
    log.save.location: log
    log.console.level: 10
    log.file.level: 10

    #MySQL Credentials
    db.host: "20.113.63.254"
    db.port: "3306"
    db.user: "s3dbuser"
    db.pwd: "gAAAAABk03F2kO04IdmnBQArBIxDW4SKTUDRRkCo-
k8adO9wnCHK1rFTkLJzGNbl2jaRa0dRjzJst6uqnq6zBsI3BDGCqV_SK
A=="
    db.name: "dpdhl_db"

    # Key to decrypt
```

```
    fernet.key:
"MmNaRbS2E9HGlEaaUmbcOQktSvfEPQASA2aUbDc3Ue8="
  volumes:
    - ./data/python-kafka-client:/app/data/
    - /var/datatransformer/log/python-kafka-client:/app/log
```

Then run following command to start the application.

```
docker compose -f docker-compose.yml up -d python-kafka-client
```

Setup & Deploy Vendororder Service

1. Create Docker Images:

   Go to the project root folder. Create a docker image of this application using following command:

```
docker build -t <username>/vendororder-service:1.2 .
```

   Here <Username> will be the user name of docker hub if you want to push this image to docker hub. Otherwise, no need to provide username if you want to run this image to your local machine.

2. Push Image:

   To push this image to docker hub, you have to use following command:

```
docker push yeasinmahi/vendororder-service:1.2
```

3. Pull Image:

   To pull image to the production/beta server please use following command to pull this image.

```
docker pull yeasinmahi/vendororder-service:1.2
```

Note: You have to have the docker install where you want to deploy the docker applications.

4. Run the application:

To start application, you need a docker compose file. So, you need to create a file named docker-compose.yml and past the following code to that file.

```
version: "3.1"
services:
  vendororder-service:
    image: "yeasinmahi/vendororder-service:1.2"
    container_name: "vendororder-service"
    restart: always
    environment:
        ## IBM config
        ibm.api.host: https://s2d-api.test.logistics.ibm.com
        ibm.auth.host: https://us-south.appid.cloud.ibm.com
        ibm.auth.url.id: 0e35b23f-70a7-4879-9e5c-4866963c4c1c
        ibm.client.id: a171ebaf-e7c8-4d27-8c63-84d2ff53b84d
        ibm.client.secret:
NTNjZmFlOGMtNGVkMy00OGE4LWIyMTctOGUwNWFmNzhiNjNh

        # trans request
        trans.api.host: http://10.10.2.4:8004
        trans.auth.host: http://10.10.2.4:8101
        trans.client.id: ebab7d00-4381-11ed-8fb6-f5122bab5d02
        trans.client.secret:
gAAAAABjO4kAAAECAwQFBgcICQoLDA0OD7JUW4ZyJY2XKSawUKU5cE
fEcgAj6jz2orL46qXNcP9mpZImtp4h6oiwES0ilVDMDA==

        # exception request
        exception.host: https://uat-transcomm.api.blesstrace.com

        ## data specific
        shipment.response.json.location: data
        shipment.response.json.isSave: 'true'
        shipment.response.pdf.location: data

        ## Log Specific
        log.save.location: log
        log.console.level: 10
        log.file.level: 10

        #MySQL Credentials
        db.host: "20.113.63.254"
```

```
        db.port: "3306"
        db.user: "s3dbuser"
        db.pwd: "gAAAAABk03F2kO04IdmnBQArBIxDW4SKTUDRRkCo-
k8adO9wnCHK1rFTkLJzGNbl2jaRa0dRjzJst6uqnq6zBsI3BDGCqV_SKA=="
        db.name: "dpdhl_db"

        # Key to decrypt
        fernet.key:
"MmNaRbS2E9HGlEaaUmbcOQktSvfEPQASA2aUbDc3Ue8="

        # Thread
        thread.interval :  20
        coninv.retry.limit :  5
        coninv.doc.retry.limit : 5
    volumes:
      - /var/datatransformer/data:/app/data/
      - /var/datatransformer/log/vendororder-service:/app/log
```

Then run following command to start the application.

```
docker compose -f docker-compose.yml up -d vendororder-service
```

Setup & Deploy Web Vendor Order Service

1. Prepare docker file:

Go to project root folder and then open terminal from the project root folder. Then go to the Src folder using following command:

```
cd src
```

Save the following docker file to a file named “Dockerfile” without any extension.

```
FROM MCR.MICROSOFT.COM/DOTNET/ASPNET:6.0 AS BASE
WORKDIR /APP
EXPOSE 80

FROM MCR.MICROSOFT.COM/DOTNET/SDK:6.0 AS BUILD

WORKDIR /SRC
COPY ["WEB.VENDORORDER/WEB.VENDORORDER.CSPROJ",
"WEB.VENDORORDER/"]
```

```
COPY ["WebBase/WebBase.csproj", "WebBase/"]
RUN dotnet restore "Web.VendorOrder/Web.VendorOrder.csproj"
COPY . .
WORKDIR "/src/Web.VendorOrder"
RUN dotnet build "Web.VendorOrder.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "Web.VendorOrder.csproj" -c Release -o
/app/publish /p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "Web.VendorOrder.dll"]
```

2. Create Docker Images:

Go to the project root folder. Create a docker image of this application using following command:

```
docker build -t <username>/dpdhl-web-vendororder:1.2 .
```

Here <Username> will be the user name of docker hub if you want to push this image to docker hub. Otherwise, no need to provide username if you want to run this image to your local machine.

3. Push Image:

To push this image to docker hub, you have to use following command:

```
docker push yeasinmahi/dpdhl-web-vendororder:1.2
```

4. Pull Image:

To pull image to the production/beta server please use following command to pull this image.

```
docker pull yeasinmahi/dpdhl-web-vendororder:1.2
```

Note: You have to have the docker install where you want to deploy the docker applications.

5. Setup config:
    - Mapper Config:
      Go to the following location: **/deployment/services/configuration/data-transformer/mapper-configs**
      Save the following mapper config named "**vo_cominv_ibm.json**"


```
{
    "MapperName": "Vendor Order Invoice from IBM",
    "Steps": [
        {
            "StepId": "001",
            "StepName": "JSONParser",
            "Config": {}
        },
        {
            "Source": [
                "001"
            ],
            "StepId": "002",
            "StepName": "DataTransformer",
            "Config": {
                "Templatepath": "configuration/mapper-
configs/templates/vo_template_cominv_ibm.json"
            }
        },
        {
            "Source": [
                "002"
            ],
            "StepId": "003",
            "StepName": "HTTPOut",
            "Config": {
                "EndPoint": "https://20.218.218.193/bless/api/v3/messages
/trusted",
                "Token": "NmZiMDEzMGMtMmU3Yi00OWU0LTliYjItMjk4MDViOTE3ZDB
l.MEQCIGclhjEKlbH7vUGOjY7Lt3edRLrf5VZqoRwamdx71rIVAiAt23ZAa4WgRxupwpaDT9H
rwXdIQHhgv+RLs3foW7H3tw==",
                "RequestFormat": "multipart/form-data",
                "CustomHeaders": {
                    "kid": "dc8778973df625c6201f9332aafe63445b8833fc2ac9b
65ddb9e00678fe8c813",
                    "resolve-receiver": false,
                    "did-enable": false,
                    "alg": "EcdsaSecp256r1Signature2019",
                    "bless-issuer": "BETA-DHL-LLP-TRF-UP",
```

```
                    "bless-message-type": "VND_ORD_IBM_COMINV",
                    "bless-subject-primary": "COURIER-FORWARDER",
                    "typ": "JWT",
                    "Created": "[DateTime]"
                },
                "IsDebug": true,
                "CustomDataObjectWrapper": "body",
                "PayloadKey": "message",
                "DataRootPath": "$.data",
                "FixedColumns": {
                    "messagetype": "VND_ORD_IBM_COMINV",
                    "org": "IBM"
                }
            }
        }
    ]
}
```

Also save the following mapper config named **“vo_cominvdoc_ibm.json”**

```
{
    "MapperName": "Vendor Order Invoice Document from IBM",
    "Steps": [
        {
            "StepId": "001",
            "StepName": "JSONParser",
            "Config": {}
        },
        {
            "Source": [
                "001"
            ],
            "StepId": "002",
            "StepName": "DataTransformer",
            "Config": {
                "Templatepath": "configuration/mapper-
configs/templates/vo_template_cominvdoc_ibm.json"
            }
        },
        {
            "Source": [
                "002"
            ],
            "StepId": "003",
            "StepName": "HTTPOut",
            "Config": {
                "EndPoint": "https://20.218.218.193/bless/api/v3/messages
/trusted",
```

```
                "Token": "NmZiMDEzMGMtMmU3Yi00OWU0LTliYjItMjk4MDViOTE3ZDB
l.MEQCIGclhjEKlbH7vUGOjY7Lt3edRLrf5VZqoRwamdx71rIVAiAt23ZAa4WgRxupwpaDT9H
rwXdIQHhgv+RLs3foW7H3tw==",
                "RequestFormat": "multipart/form-data",
                "CustomHeaders": {
                    "kid": "dc8778973df625c6201f9332aafe63445b8833fc2ac9b
65ddb9e00678fe8c813",
                    "resolve-receiver": false,
                    "did-enable": false,
                    "alg": "EcdsaSecp256r1Signature2019",
                    "bless-issuer": "BETA-DHL-LLP-TRF-UP",
                    "bless-message-type": "VND_ORD_IBM_COMINV_DOC",
                    "bless-subject-primary": "COURIER-FORWARDER",
                    "typ": "JWT",
                    "Created": "[DateTime]"
                },
                "IsDebug": true,
                "CustomDataObjectWrapper": "body",
                "PayloadKey": "message",
                "DataRootPath": "$.data",
                "FixedColumns": {
                    "messagetype": "VND_ORD_IBM_COMINV_DOC",
                    "org": "IBM"
                }
            }
        }
    ]
}
```

- Template:
  Go to the following location: **/deployment/services/configuration/data-transformer/mapper-configs/templates**
  Save the following template named "**vo_template_cominv_ibm.json**"

```
{
    "data": {
        "org": "IBM",
        "messagetype": "VND_ORD_IBM_COMINV",
        "shipmenttrackingnumber": "#valueof($.shipment)",
        "invoicenumber": "#valueof($.invoice)",
        "payload": "#valueof($.payload)"
    }
}
```

  Also save the following template named **"vo_template_cominvdoc_ibm.json"**

```
{
    "data": {
        "shipmenttrackingnumber": "#valueof($.shipment)",
        "invoicenumber": "#valueof($.invoice)",
```

```
            "filename": "#valueof($.file)"
        }
    }
```

6. Run the application:

To start application, you need a docker compose file. So, you need to create a file named docker-compose.yml and past the following code to that file.

```
version: "3.1"
services:
  web-vendororder:
    image: s3innovatesg/dpdhl-web-vendororder:1.0
    hostname: dpdhl-web-vendororder
    container_name: dpdhl-web-vendororder
    depends_on:
      - data-transformer
    restart: unless-stopped
    logging:
      driver: "local"
      options:
        max-file: "10"
        max-size: "20m"
    ports:
      - "8004:80"
    environment:
      JWT__VALIDITY: 30
      CONNECTIONSTRINGS__DEFAULT:
"Server=mysql;Port=3306;Database=dpdhl_db;Uid=s3dbuser;Pwd=gAAAA
ABjoncgkMv-Q0g3PREa3-DfPI1Ra0kP1k74dKE6FIVeE2mCA-
iiFUCWShICy2rEvZjBLf4NqYv6Hnw6KsHrNfbUqd0p0A==;"
      REST__DATATRANSFORMER__URL: "http://data-transformer:80"
    extra_hosts:
      mysql: 10.0.0.7
    volumes:
      - ./var/datatransformer/log/dpdhl-web-vendororder:/app/log
```

Then run following command to start the application.

```
docker compose -f docker-compose.yml up -d web-vendororder
```