Batch to API - Design for Resiliency Using Oracle Integration

Architecture, Integration
July 19, 2023
Peter Obert

Oracle Technology Engineering – Application Integration Specialist

Contents

Batch to API - Design for Resiliency Using Oracle Integration 1

Oracle Technology Engineering – Application Integration Specialist 1

Batch to API - Design for Resiliency Using Oracle Integration 3

Design Batch Integrations 3

Design Components 5

Post Requests 5

Load Batch Requests into Parking Lot Table 6

Dispatch Requests According to Schedule 6

File row to API call 7

Handle Failed Batch Rows 8

Payload Correction 9

Batch to API - Design for Resiliency Using Oracle Integration

Use these best practices when designing your resilient batch integrations.

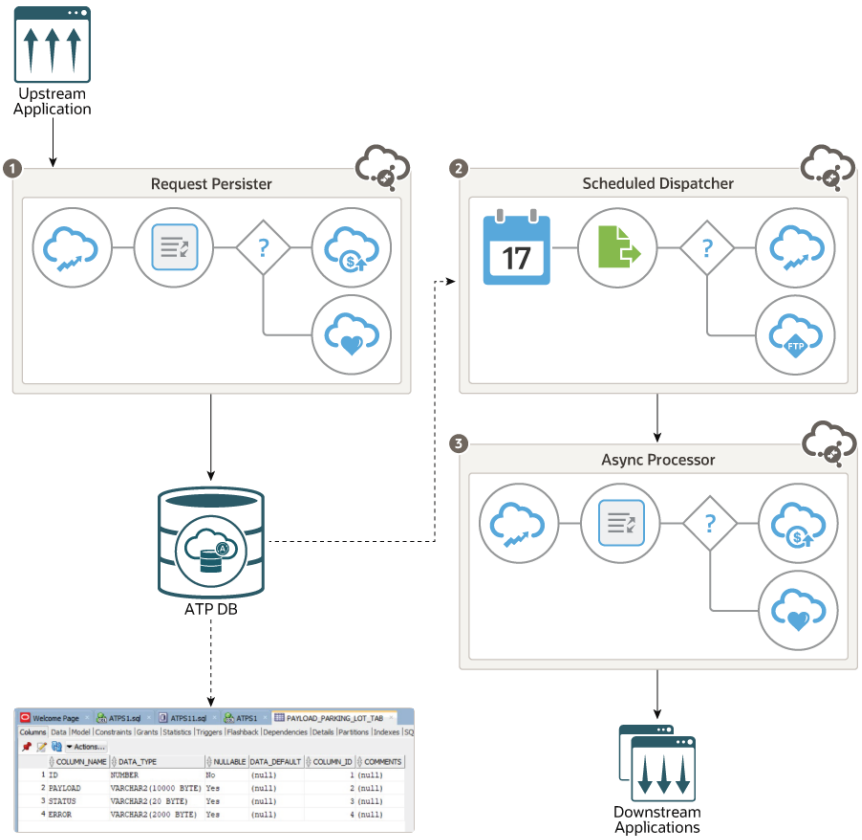
Design Batch Integrations

Here's a basic batch integration flow that receives batch requests (e.g., CSV file) from an upstream application. File is read from the source directory, parsed, validated, and processed as set of API calls to downstream application.

There may be a case where the downstream application becomes unresponsive, or data are not passing business criteria or other processing problems. These API requests will not be acknowledged by the downstream system.

Let's take an example of regularly creating aggregated CSV file of changed Employee Bank Connections in Oracle HCM cloud. The file must be received by an Integration adapter. You should be able to dynamically throttle the number of running batches and requests produced by HCM or other upstream system and track the status of the requests on the row level and resubmit any failed requests.

For this solution, three integrations and an Autonomous Transaction Processing database are shown. The implementation of the parking lot can be done using various storage technologies like database or Coherence. However, we're using an Autonomous Transaction Processing database table for simplicity.



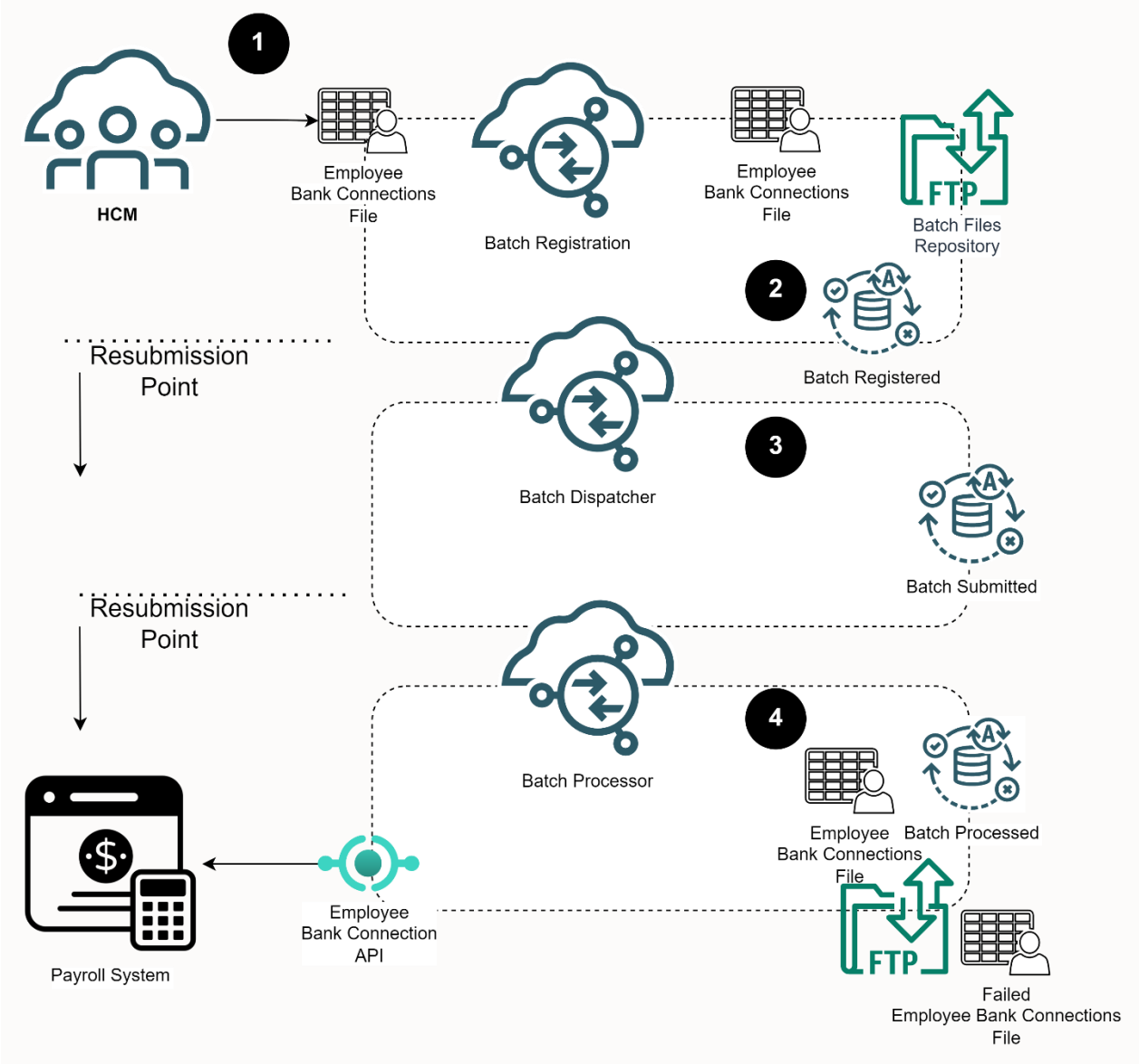
In the image shown, when the upstream application produces the CSV file request. File delivery can trigger the flow if OIC agent is used or scheduled process which scans the directory (Efficient File Handler Pattern with OIC Scheduler for SaaS Integration).

The Batch Request Persister integration moves the file to processing directories and registers the batch to the database. In the database, the parking lot pattern stores batch request metadata and status information and will process the batch requests based on the order they come in. Each batch request is parked in the storage for x minutes (parking time) so the integration flow has a chance to throttle the number of batches processed concurrently.

An orchestrated Schedule Dispatcher integration is triggered by a schedule. You can schedule this integration run the parked batch process from the database at a date and time of your choosing. You can also define the frequency of the integration. Schedule dispatcher updates status of the chosen batch requests to submitted for processing state.

Schedule Dispatcher hands the batch requests over to the Asynchronous batch processor integration.

Asynchronous processor integration will process the incoming file into the single rows and payload requests and send it to the downstream application.



Design Components

The high-level design has three integrations and a database. We're taking employee bank account HCM->Payroll synchronization as an example, but, it could be any batch and business objects available through HCM Extract or BIP Report or exposed by any Oracle SaaS REST API.

Post Requests

The Batch Request Persister is scheduled or file adapter triggered integration, which can be called by/submitted on event through OIC REST API on demand and handles incoming batch request.

This Batch Request Persister integration downloads HCM employee bank connections file and stores it in the local OIC fileserver for further processing. Batch Request Persister can be extended with file parsing/validation and persisting rows separately for e.g. data quality check, duplicate business entities occurrence and so on.

Batch Request Details are inserted into the Autonomous Transaction Processing database immediately. Batch Request Persister can read its configuration for the batch type (group_type) and batch business identifier (group id). The Batch_ID is generated in sequence in which batches are coming and the barch request metadata are persisted into the parking lot table for subsequent processing.

Load Batch Requests into Parking Lot Table

The Autonomous Transaction Processing database here holds the parking lot table where all received requests are parked before processing. For simplicity, a simple table is shown to persist the payload for single payload style or persisting file reference for the batch type requests. It contains request status and any request processing error information.

For the batch requests type processing only file reference is populated and updated during the processing stages.

Proposed solution keeps capability to store JSON/SOAP message request payload which is entirely stored in the parking lot table as a string.

```
create sequence "ADMIN"."sqname_id_seq" start with 1 increment by 1 nocache;
drop table "ADMIN"."PAYLOAD_PARKING_LOT_TAB";
CREATE TABLE "ADMIN"."PAYLOAD_PARKING_LOT_TAB"
   (    "ID" NUMBER(*,0) DEFAULT "ADMIN"."sqname_id_seq".nextval, -- BATCHID
    "GRPID" VARCHAR2(40 BYTE) COLLATE "USING_NLS_COMP",
    "GRPTYPE" VARCHAR2(40 BYTE) COLLATE "USING_NLS_COMP",
    "GRPSEQ" NUMBER(*,0),
    "PAYLOAD_CLOB" CLOB COLLATE "USING_NLS_COMP",
    "PAYLOAD" VARCHAR2(10000 BYTE) COLLATE "USING_NLS_COMP",
    "FILENAME" VARCHAR2(40 BYTE) COLLATE "USING_NLS_COMP",
    "DIRECTORY" VARCHAR2(200 BYTE) COLLATE "USING_NLS_COMP",
    "STATUS" VARCHAR2(20 BYTE) COLLATE "USING_NLS_COMP",
        -- NEW - new request/batch
        -- SUBMITTED - batch processing triggered
        -- PROCESSING - batch contitions passed and processing started
        -- FAILED - request/batch processing failed
        -- ERROR_RETRY - request/batch ready to be re-processed
    "CREATIONTIME" TIMESTAMP(3) NOT NULL,
    "STATUSTIME"   TIMESTAMP(3) NOT NULL,
    "ERROR" VARCHAR2(2000 BYTE) COLLATE "USING_NLS_COMP"
   )  DEFAULT COLLATION "USING_NLS_COMP" ;
```

Dispatch Requests According to Schedule

Scheduled Integration is scheduled to run at the required frequency. In every run it fetches a configured number of requests and loops through them dispatching each request to an Async Batch Processor integration for processing.

Note: batch type (group_type) and batch business identifier (group_id) can be filtering parameters for the implemented dispatcher).

You can configure to fetch several batch requests as a scheduled parameter to throttle or accelerate the request processing, and also to dynamically change the value. For example, you can set up a table in such a way that the requests from parking lot table are fetched based on status of requests. You can fetch **`NEW`** and **`ERROR_RETRY`** status requests and pass on for processing.

This Scheduled Batch Dispatcher then loops through the fetched number of batch requests and hands off each request to the Asynchronous Batch processor for employee bank connection synchronization.

Make sure that the Scheduler (parent) flow calls a one-way Asynchronous Integration (child) flow. The Async Batch Processor does not return any response and loop through the rest of the registered batch requests and dispatch them.

If we plan to run **batch rows processing in parallel**, then Async Batch Processor should be implemented as scheduled orchestration and Dispatcher flow should trigger OIC integration as **`RUN NOW`**.

File row to API call

Asynchronous Batch Processor integration will process the batch file requests from the Scheduled Dispatcher. It reads the file using the schema and sends API calls to the downstream application.

The asynchronous batch processor exposes a REST interface. It is important that this integration is modeled as one-way asynchronous flow.

Since the asynchronous flow does the actual employee bank connection synchronization, it will be responsible to update the batch processing request status and **Batch Processing Statistics** for the processed rows, succeeded rows, failed rows. After the processing every batch file row it updates the status of the processing in the parking lot table. After a successful batch processing for employee bank connection row in the file the `STATUS` column in parking lot table is updated to `PROCESSED`.

```
--drop table "ADMIN"."BATCH_STATISTICS";
CREATE TABLE "ADMIN"."BATCH_STATISTICS"
   (
    "BATCHID" INT NOT NULL,
    "GRPID" VARCHAR2(40 BYTE),
    "GRPTYPE" VARCHAR2(40 BYTE),
      "GRPSEQ" INT,
    "TOTAL_ROWS" INT,
    "PROCESSED_ROWS" INT,
    "SUCCEEDED_ROWS" INT,
    "FAILED_ROWS" INT,
    "CREATIONTIME" TIMESTAMP(3) NOT NULL,
    "STATUSTIME"   TIMESTAMP(3) NOT NULL,
    "STATUS" VARCHAR2(20 BYTE) COLLATE "USING_NLS_COMP",
    "ERROR" VARCHAR2(2000 BYTE) COLLATE "USING_NLS_COMP"
 );
CREATE UNIQUE INDEX "ADMIN"."BATCH_STATISTICS_PK_ID" ON "ADMIN"."BATCH_STATISTICS" ("BATCHID");
ALTER TABLE "ADMIN"."BATCH_STATISTICS" ADD CONSTRAINT "BATCHID" PRIMARY KEY ("BATCHID");
```

Handle Failed Batch Rows

Resubmission of failed requests can be controlled from the payload error table.

A scope level error handler handles any faults during the asynchronous batch file processing and single payroll API call. The error handler creates record and persists the API call payload. The status of the payload is ERRORED for any errors.

The reason and error details are also updated from the error payload of the API call in the scope error handler. This is useful for determining if the request can be resubmitted later.

```
--drop table "ADMIN"."PAYLOAD_ERRORS_TAB";
create sequence "ADMIN"."error_id_seq" start with 1 increment by 1 nocache;
CREATE TABLE "ADMIN"."PAYLOAD_ERRORS_TAB"
   ( "ERRID" INT DEFAULT "ADMIN"."error_id_seq".nextval,
    "BATCHID" INT NOT NULL,
    "GRPID" VARCHAR2(40 BYTE),
    "GRPTYPE" VARCHAR2(40 BYTE),
      "GRPSEQ" INT,
    "PAYLOAD_CLOB" CLOB,
    "PAYLOAD" VARCHAR2(10000 BYTE) COLLATE "USING_NLS_COMP",
    "STATUS" VARCHAR2(20 BYTE) COLLATE "USING_NLS_COMP",
       -- NEW after registering by initial BATCH
       -- READY after fixing the payload
       -- Whatewer else after failure
    "ERROR" VARCHAR2(2000 BYTE) COLLATE "USING_NLS_COMP",
      "CREATIONTIME" TIMESTAMP(3) NOT NULL,
    "STATUSTIME"   TIMESTAMP(3) NOT NULL
   )  ;
CREATE UNIQUE INDEX "ADMIN"."PKERR_ID" ON "ADMIN"."PAYLOAD_ERRORS_TAB" ("ERRID");
ALTER TABLE "ADMIN"."PAYLOAD_ERRORS_TAB" ADD CONSTRAINT "PKERR_ID" PRIMARY KEY ("ERRID");
```

To re-play failed batch rows as the new batch, scope error handler can create new batch file and append every failed row into the error file.

You can also have error notification emails sent out to integration administrators after the finishing all the batch rows processing with the batch processing statistics.

If any error is captured during the all the batch rows processing, then Asynchronous batch processor should set the batch processing status to **`ERRORED`** status in the parking lot table.

Resolution on a row level in the error payloads table is done the way that the payload is fixed and updated to **`READY`** status. These rows can be picked by the specialized Scheduled Dispatcher Payroll Errors in the schedule for re-processing due to the selection criteria of the payload errors table in Autonomous database.

- The Asynchronous Processor integration's fault handler can set the status to directly, so every failure gets resubmitted automatically in the next schedule.

Re-play original file or Error file is possible by updating parking lot table status to **`RETRY`** and file reference and directory values to error file location. Business conditions must be set/fixed prior re-play/re-submit the batch or error file batch.

Payload Correction

Storing the employee bank connection payload in the error payloads table has given us a way to correct the payload of data errors prior to resubmission. Update the payload and set status column to **`READY`** to resubmit a request with corrected payload using specialized Scheduled Dispatcher Payroll Errors.