Case Study Design

Contents

[High level design document 2](#_Toc70966650)

[ 2](#_Toc70966651)

[Batch File Processing Flow 2](#_Toc70966652)

[Ingress File Layer 2](#_Toc70966653)

[Transaction Listener 3](#_Toc70966654)

[Validator service 3](#_Toc70966655)

[Downstream Routing Service 4](#_Toc70966656)

[Persistence Service 4](#_Toc70966657)

[Discovery service 5](#_Toc70966658)

[Retrieval Flow 5](#_Toc70966659)

[End to End test cases/ GUI / reporting application 5](#_Toc70966660)

[Application Gateway 5](#_Toc70966661)

[Persistence Service 5](#_Toc70966662)

[Best Practices 5](#_Toc70966663)

[Below things are not implemented 6](#_Toc70966664)

# High level design document

# 

|  |
| --- |
|  |

# Batch File Processing Flow

Files will be placed in staging folder

staging folder = [C://Mahesh//code//hsbc\_case\_study//staging]

File Name format = input\_\*

When ReadyFile.txt file is copied along with file specified in above name format it will be picked for processing and moved to completed folder once done.

Some of the major components in file processing

## Ingress File Layer

This is apache camel application. This application will chop file and each line is send to active mq as one transaction. Fields of the line are separted by “,”. Other separator can also be supported but we have to add more convertor

We can create multiple instances of this application. We have to pass staging folder and file name format as command line arguments.

|  |
| --- |
|  |

There is no logic involved here hence ingesting to queue will be very fast.

Message are sent to: inbound.queue

## Transaction Listener

This spring boot application.

Listens to queue: inbound.queue

Error Queue: error.queue

If successful messages are sent to: validation.queue

It will persist transaction in transaction table as Received by invoking load balanced URL of persistence servce. In case of any failure it will mark message as Failed or/and it will send message to error queue.

Any message max will be attempted max 3 times in order to avoid poison message problem.

This will also register to service discovery service.

This service is not dependent on any other service except persistence service. Hence it increases availability of the system. This is also can be scaled independently.

## Validator service

This spring boot application.

Listens to queue: validation.queue

Error Queue: error.queue

If successful messages are sent to: routing.queue

It will validate instrument id form preconfigured set of instrument id in hash map.

In ideal case scenario, wanted to keep list of instrument ids in distributed cache. Some service will interact with golden sources to keep cache up to date.

But because of time constraint. This functionality not implemented.

Once validation is completed it will invoke load balanced URL of persistence service and marked status as validated. In case of any failure it will mark message as Failed or/and it will send message to error queue.

This service is not dependent on any other service except persistence service. Hence it increases availability of the system. This is also can be scaled independently.

## Downstream Routing Service

This spring boot application.

Listens to queue: validation.queue

Error Queue: error.queue

If successful messages are sent to: outbound.queue1, outbound.queue2, outbound.queue3 Based on routing rule.

It will route message to downstream system based on preconfigured set of rules in hash map.

In ideal case scenario, wanted to keep list of routing rules in. Some service will interact with golden sources to keep cache up to date.

But because of time constraint. This functionality not implemented.

Once routing is completed it will invoke load balanced URL of persistence service and marked status as completed. In case of any failure it will mark message as Failed or/and it will send message to error queue.

This service is not dependent on any other service except persistence service. Hence it increases availability of the system. This is also can be scaled independently.

## Persistence Service

This service spring boot application. It handles all the use cases of persistence.

Why this service is synchronous?

The reason for keeping for this service synchronous is that we may have multiple instances of this service running. It may possible that two persistence request of same transaction can be picked parallel, in such case system may not worked correctly.

It provides uri for persistence

http://<base-url>/app/transaction/

Http method

Post: insertion of transaction

Put: updation of transaction

Get: fetch of transaction

## Discovery service

This is standard discovery service. It is used to improve availability of system by providing fault tolerance of some instances of particular service. It also aid in client side load balancing.

# Retrieval Flow

Major components in retrival flow

## End to End test cases/ GUI / reporting application

These are end user of application may want data processed by application. Status of particular file processed etc.

## Application Gateway

This will provide single point of access to all application in the system. This will interact with discovery service and will used load balanced url of services for better availability and fault tolerance.

## Persistence Service

This will provide data required by end client.

# Best Practices

* Design created adheres to microservices IDEALS principle

Inter segregation principle: N/A applicable in this case but can be easily done by adding endpoint specific to client of application

Deplorability on You: Not applicable in this case

Event Driven: Messaging is used to create event driven architecture. It increase availability.

Availability over consistency: N/A

Loosely Coupled: Entire application is loosely coupled. Except persistence system no other other service depends on other service. Even if some services are not available application can continue till queue has space. Even though other services are dependent on persistence service since it is routed through load balanced URL we have increased fault tolerance.

Single Responsibility: It appears that micro services are appropriately sized but it is more of subjective measure.

* Database is not shared [Validation service and instrumentation service can have its own schema/ database to store validation and routing rules]
* Repositories are kept separate except domain entities nothing is shared.
* Poison message use case handle

# Below things are not implemented

Because of time constraint some of the important things are missed

* Security: Both application level security [Authentication/Authorization] and transport layer security is not implemented [tls]
* Distributed transaction management is not done
* Exception handling is primitive