



**High Level Design**

**For**

**Fat Face - Demandware Integration**

27-Mar-2017

WHISHWORKS

www.whishworks.com

WW/PD/REC/HLD/1/LON01/FFLL1

Status: Restricted

Prepared By: rupesh.sinha@whishworks.com

*© WHISHWORKS 2017. All rights reserved.*

*No part of this document may be reproduced or transmitted in any form or by any means electronic or mechanical including photocopying and recording or by any information storage or retrieval system except as may be expressly permitted by WHISHWORKS*

**DISTRIBUTION LIST**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Name** | **Contact Details** | **Purpose** |
| 1 | Mark Jones | [mark.jones@fatface.com](mailto:mark.jones@fatface.com) | Approver |
| 2 | Samin Sarkar | [samin.sarkar@fatface.com](mailto:samin.sarkar@fatface.com) | Reviewer |
| 3 | Gary Bolton | [gary.bolton@fatface.com](mailto:gary.bolton@fatface.com) | Informed |
| 4 | Rob Mabey | [rob.mabey@whishworks.com](mailto:rob.maybey@whishworks.com) | Informed |
| 5 | Sreekanth Cherukuri | [sreekanth.cherukuri@whishworks.com](mailto:sreekanth.cherukuri@whishworks.com) | Reviewer |
| 6 | Srikanth Abdullapur | [srikanth.abdullapur@whishworks.com](mailto:srikanth.abdullapur@whishworks.com) | Reviewer |
| 7 | Satyamurty Kodukulla | [satyamurty.Kodukulla@whishworks.com](mailto:satyamurty.Kodukulla@whishworks.com) | Informed |
| 8 | Sudarsana Sangaraju | [sudarsana.sangaraju@whishworks.com](mailto:sudarsana.sangaraju@whishworks.com) | Informed |
| 9 | Imam Shareef | imam.shareef@whishworks.com | Reviewer |
| 10 | Shruthi Thalari | [shruthi.thalari@whishworks.com](mailto:shruthi.thalari@whishworks.com) | Reviewer |

**DOCUMENT CONTROL**

|  |  |  |  |
| --- | --- | --- | --- |
| **Prepared By** | **Reviewed By** | **Approved By** | **Approved Version** |
| Rupesh Sinha | Sreekanth A | - | - |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**VERSION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 14-Mar-2017 | 0.1 | Initial Draft | Rupesh Sinha |
| 15-Mar-2017 | 0.2 | Updated based on Sreekanth Abdullapur’s comments | Rupesh Sinha |
| 16.Mar-2017 | 0.3 | Updated as per Sreekanth Cherukuri’s comments | Rupesh Sinha |
| 16-Mar-2017 | 0.4 | Updated testing section as per Imam Shareef comments | Rupesh Sinha |
| 17-Mar-2017 | 0.5 | Added comments around batch integration approach | Rupesh Sinha |
| 17-Mar-2017 | 1.0 | First draft to Customer – version changed to V1.0 | Rupesh Sinha |
| 17-Mar-2017 | 1.1 | Initial comments added based on HLD review meeting with FATFACE | Rupesh Sinha |
| 27-Mar-2017 | 1.2 | Updated based on review comments from FATFACE and WHISHWORKS TDA | Rupesh Sinha |
| 27-Mar-2017 | 1.3 | Moved data migration interfaces into in scope section and updated link to reference documents | Rupesh Sinha |
| 13-Apr-2017 | 1.4 | Addressed review comments provided by FatFace | Rupesh Sinha |
| 20-Apr-2017 | 1.5 | Updated based on PIM, JDA and Prologic discovery outcome | Rupesh Sinha |
| 07-Aug-2017 | 2.0 | Updated HLD based on new deployment model based on 4 vCore licensing and also include requirements and scope changes | Rupesh Sinha |
| 22-Aug-2017 | 2.1 | Address review comments from WW TDA | Rupesh Sinha |
| 20-Oct-2017 | 2.2 | Updated HLD to include CCMP application in scope for transactional emails | Rupesh Sinha |
| 24-Oct-2017 | 2.3 | Updated figure 2, licensing under 6.1.1.3. Added comments around SFTP server sizing and encryption method used for the properties file | Rupesh Sinha |

**REFERENCES**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Document/Artefact** | **Version** | **Date** |
| 1 | [Best Practices \_ Guidelines for Mule ESB.pdf](https://fatface.atlassian.net/wiki/download/attachments/94477511/Best%20Practices%20&%20Guidelines%20for%20Mule%20ESB-020317.pdf?api=v2) | 1.0 | 17/03/2017 |
| 2 | [Best Practices \_ Guidelines for API development.pdf](https://fatface.atlassian.net/wiki/download/attachments/94477511/Best%20Practices%20&%20Guidelines%20for%20API%20development-020317.pdf?api=v2) | 1.0 | 17/03/2017 |
| 3 | [AnypointNamingConventions.pdf](https://fatface.atlassian.net/wiki/download/attachments/94477511/AnypointNamingConventions-020317.pdf?api=v2) | 1.0 | 17/03/2017 |
| 4 |  |  |  |

**DEFINITIONS & ACRONYMS**

|  |  |  |
| --- | --- | --- |
| **#** | **Acronym** | **Expanded Form** |
| 1 | TBC | To be considered |
| 2 | SOA | Service Oriented Architecture |
| 3 | HLD | High Level Design |
| 4 | POS | Point of Sale |
| 5 | TBD | To be determined |
| 6 | ECOM | E-commerce |
| 7 | EPOS | Electronic Point of Sale |
| 8 | VPC | Virtual Private Cloud |
| 9 | VPN | Virtual Private Network |
| 10 | DW | Demandware |
| 11 | JMS | Java Messaging Service |
| 12 | AMQ | Anypoint Messaging Queue |

Table of Contents

[1. Executive Summary/Overview 8](#_Toc496628780)

[2. Solution Context 9](#_Toc496628781)

[2.1.1.1 Objectives 9](#_Toc496628784)

[2.1.1.2 Critical Success Factors 9](#_Toc496628785)

[2.1.1.3 High Level Business Requirements 9](#_Toc496628786)

[2.1.1.4 Exclusions 10](#_Toc496628787)

[2.1.1.5 Risks 10](#_Toc496628788)

[2.1.1.6 In Scope 10](#_Toc496628789)

[2.1.1.7 Out of Scope 11](#_Toc496628790)

[2.1.1.8 Assumptions 12](#_Toc496628791)

[2.1.1.9 Dependencies 12](#_Toc496628792)

[2.1.1.10 Outstanding Issues 13](#_Toc496628793)

[2.1.1.11 Terminology 13](#_Toc496628794)

[3. Technical Architecture Overview 14](#_Toc496628795)

[3.1.1.1 API Architecture Components 15](#_Toc496628798)

[3.1.1.2 FATFACE Architecture 16](#_Toc496628799)

[3.1.1.3 Components of Application 18](#_Toc496628800)

[3.1.1.4 ESB Components 18](#_Toc496628801)

[3.1.1.5 Database Components 19](#_Toc496628802)

[3.1.1.6 Other Components 19](#_Toc496628803)

[4. Application Design 22](#_Toc496628804)

[4.1.1.1 Architecture Views 22](#_Toc496628807)

[4.1.1.2 Use Case View 23](#_Toc496628808)

[4.1.1.3 Update Marketing Preferences 23](#_Toc496628809)

[4.1.1.4 Create Order in Demandware 24](#_Toc496628810)

[4.1.1.5 Export Order to JDA 24](#_Toc496628811)

[4.1.1.6 Order Update in JDA 24](#_Toc496628812)

[4.1.1.7 Order Update in CIMS 24](#_Toc496628813)

[4.1.1.8 Product Update in CIMS 24](#_Toc496628814)

[4.1.1.9 Store Opening Hours Update 25](#_Toc496628815)

[4.1.1.10 Inventory Update in JDA 25](#_Toc496628816)

[4.1.1.11 Inventory Update in CIMS 25](#_Toc496628817)

[4.1.1.12 Pricebook Update in CIMS 25](#_Toc496628818)

[4.1.1.13 Payment Capture Request 25](#_Toc496628819)

[4.1.1.14 Refund Request 26](#_Toc496628820)

[4.1.1.15 Logical View 26](#_Toc496628821)

[4.1.1.16 Implementation View 27](#_Toc496628822)

[4.1.1.17 Process View 31](#_Toc496628823)

[4.1.1.18 Deployment View 34](#_Toc496628824)

[4.1.1.19 Design Considerations 36](#_Toc496628825)

[4.1.1.20 Design Solutions 37](#_Toc496628826)

[4.1.1.21 Mule Connectors 41](#_Toc496628827)

[4.1.1.22 Mule Message Processors 41](#_Toc496628828)

[4.1.1.23 Data Model 42](#_Toc496628829)

[4.1.1.24 Protocols and Formats 42](#_Toc496628830)

[4.1.1.25 NFR Solutions 42](#_Toc496628831)

[4.1.1.26 Availability 42](#_Toc496628832)

[4.1.1.27 Scalability 42](#_Toc496628833)

[4.1.1.28 Reliability 43](#_Toc496628834)

[4.1.1.29 Security 43](#_Toc496628835)

[4.1.1.30 Auditing 43](#_Toc496628836)

[4.1.1.31 Performance 44](#_Toc496628837)

[4.1.1.32 Exception and Error Handling and Alerting 44](#_Toc496628838)

[4.1.1.33 Application Monitoring 46](#_Toc496628839)

[4.1.1.34 Batch Processing and Scheduling 46](#_Toc496628840)

[4.1.1.35 Build and Deployment Solution 48](#_Toc496628841)

[4.1.1.36 Build Management 48](#_Toc496628842)

[4.1.1.37 Continuous Integration 48](#_Toc496628843)

[4.1.1.38 Tools and Technologies 48](#_Toc496628844)

[4.1.1.39 Development Tools 48](#_Toc496628845)

[4.1.1.40 Technologies 48](#_Toc496628846)

[4.1.1.41 Environment 48](#_Toc496628847)

[4.1.1.42 Server Inventories 49](#_Toc496628848)

[5. Testing 52](#_Toc496628849)

[5.1.1.1 Testing Approach 52](#_Toc496628853)

[5.1.1.2 Functional Testing 52](#_Toc496628854)

[5.1.1.3 Non-Functional testing 52](#_Toc496628855)

[5.1.1.4 Testing Tools 52](#_Toc496628856)

[5.1.1.5 Mocking frameworks 52](#_Toc496628857)

[6. Licensing and Support 53](#_Toc496628858)

[6.1.1.1 Support Ownership 53](#_Toc496628859)

[6.1.1.2 Service Levels 53](#_Toc496628860)

[6.1.1.3 Licensing 53](#_Toc496628861)

[6.1.1.4 Environment Management 53](#_Toc496628862)

Table of Figures

[Figure 1. FATFACE AS IS Architecture 8](#_Toc496628863)

[Figure 2. FATFACE Demandware integration scope 9](#_Toc496628864)

[Figure 3. High Level Mule Cloud Architecture. 10](#_Toc496628865)

[Figure 4. Technical Architecture Overview 14](#_Toc496628866)

[Figure 5. FATFACE Architecture 17](#_Toc496628867)

[Figure 6. Common Framework 20](#_Toc496628868)

[Figure 7. Architecture View 22](#_Toc496628869)

[Figure 8. IPsec VPN connection 23](#_Toc496628870)

[Figure 9. VPC Peering connection 23](#_Toc496628871)

[Figure 10. Logical View 26](#_Toc496628872)

[Figure 11. X-Layer: Pattern-1: X-API calling S-API 28](#_Toc496628873)

[Figure 12. X-Layer: Pattern-2: X-API calling P-API and P-API calling S-API 28](#_Toc496628874)

[Figure 13. P-Layer: Pattern-1: P-API batch request to S-API single request 29](#_Toc496628875)

[Figure 14. P-Layer: Pattern-3: Data pulled by calling a 3rd party web service 29](#_Toc496628876)

[Figure 15. S-Layer: Pattern-1: Pulling data from 3rd party web service 30](#_Toc496628877)

[Figure 16. S-Layer: Pattern-2: Pushing data by calling a 3rd party web service 30](#_Toc496628878)

[Figure 17. S-Layer: Pattern-3: Pushing data on SFTP server 31](#_Toc496628879)

[Figure 18. S-Layer: Pattern-4: Pulling data from SFTP server 31](#_Toc496628880)

[Figure 19. Deployment View 34](#_Toc496628881)

[Figure 20. Mule API Project structure 40](#_Toc496628882)

[Figure 21. Continuous Integration 48](#_Toc496628883)

# Executive Summary/Overview

As shown in the figure 1, FATFACE currently runs their e-commerce website on Venda e-commerce platform. They use Prologic CIMS to manage order coming from Venda and their bricks and mortar stores. FATFACE maintains their customer master data in a system known as XCM. They have various SQL server databases for performing data transformation & data transfer from one system to the other. These databases are also used for various reporting requirements. Each of these systems are integrated using point-to-point integration.

The current Venda e-commerce platform does not support multicurrency and multi geography impacting FATFACE vision of the e-commerce platform. Point-to-point integration between the systems does not offer enough flexibility to deliver on fast growing business demands.

FATFACE decided to migrate their e-commerce platform to Demandware Salesforce commerce cloud that supports multicurrency and multi geography and delivers the speed, agility, innovation and superior economics required to master the new retail reality of constant and unpredictable change.

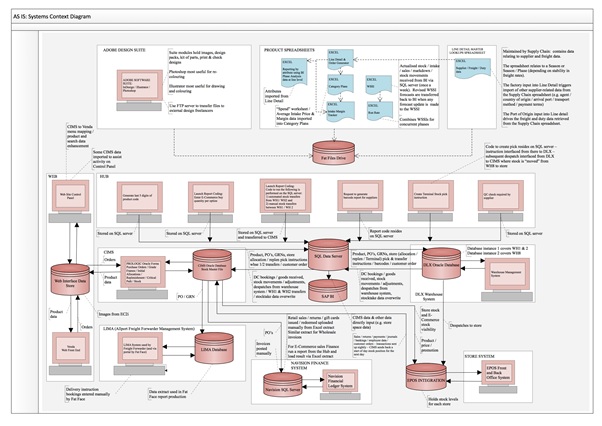


Figure . FATFACE AS IS Architecture

FATFACE engaged with WHISHWORKS to address their need to remove point-to-point integration with existing legacy systems, and standardise some of the old process involved while transforming and posting files as dictated by their legacy system (Venda).

FATFACE has chosen MuleSoft Anypoint platform as their integration platform to integrate their various systems as part of their e-commerce website migration from Venda to Demandware Salesforce e-commerce cloud.

# Solution Context



## Objectives

The purpose of this document is to provide a high level design for integrating FATFACE legacy systems with Demandware using the MuleSoft Anypoint platform. The high level design focuses on MuleSoft Anypoint architecture and reduces point-to-point integration, unlocking data and capability between systems. Following picture shows the overall integration scope for the e-commerce project.

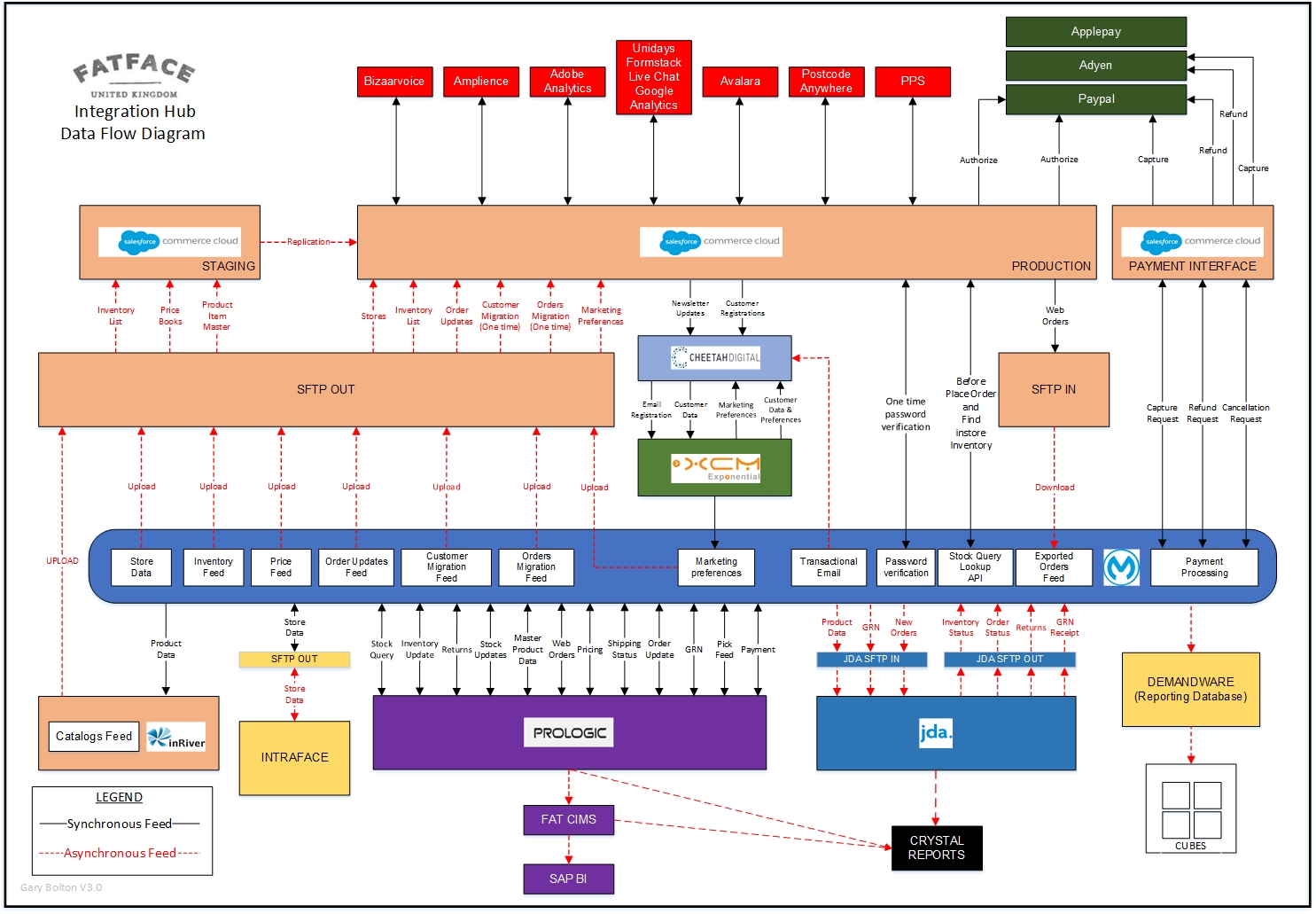
****

Figure . FATFACE Demandware integration scope

## Critical Success Factors

* Reducing point to point integrations that currently exist within the legacy systems and complexity caused by them.
* Develop common framework components.
* Expose the APIs to the trade partners using Anypoint API manager developer portal.
* Design, build, test and implement the Anypoint API interfaces for product, customer, order, inventory, price book, payment settlement and order refund.

## High Level Business Requirements

FATFACE constantly update their website by providing details via files to a legacy system Venda. Venda is currently being replaced by Demandware having much more capabilities, functionalities and less complexities involved in doing the same. Currently Venda (legacy system) consumes files in PSV format. Demandware on the other hand consumes or requires file to be provided in a predefined XML format.

MuleSoft Anypoint Platform provides a single unified platform for all connectivity needs (Legacy, SaaS and APIs) for FATFACE, beyond the initial Demandware integration requirement. The platform enables an API-Led approach to connectivity, which utilises reusable componentised architecture and unlocks data assets while providing the layers of abstraction and control between mission critical legacy systems and applications.

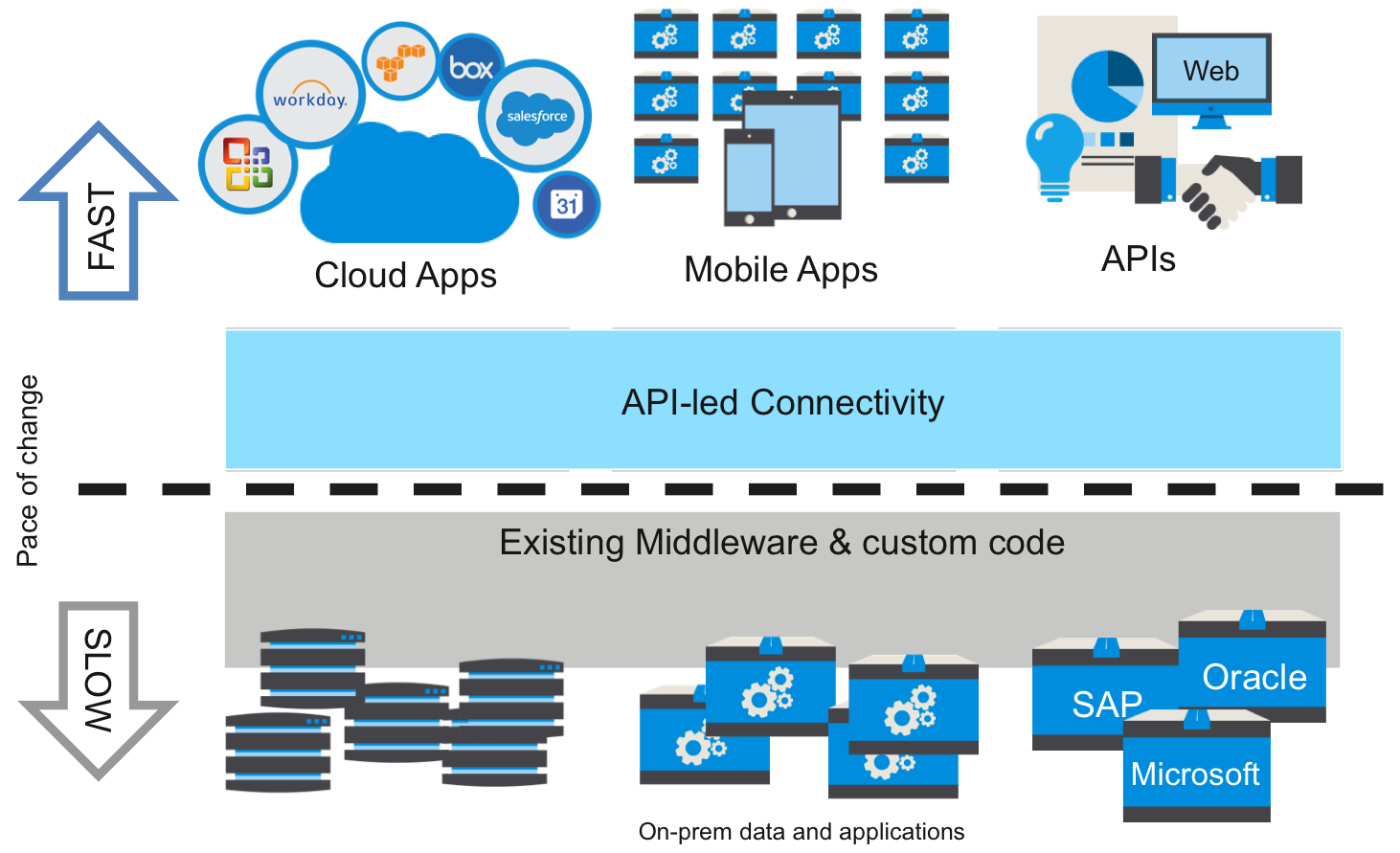


Figure . High Level Mule Cloud Architecture.

## Exclusions

* Purging of the files on the SFTP server is excluded in the design. This should be managed separately by FATFACE. This can be setup as cron job on the SFTP server. WHISHWORKS recommends to purge any files older than 30 days.

## Risks

* Timely availability of respective tools for development, SFTP folder on cloud, GitHub, Jenkins, MS-SQL DB etc.
* Verified and agreed detailed field mappings between Demandware and various backend systems.
* The maximum file size provided by FATFACE for the current integration is 2.2 MB for order update. File size for other interfaces are less than 1 MB. For the new integration solution, Demandware export file size will not export 3MB size. As it’s a cloud based application, as well as the file size to be consumed is much less, the current implementation/architecture proposed will have or involve in-memory file processing and storage of temporary files. In case if the file sizes are too big then a separate approach would have to be devised, which in turn will end up having a network connectivity and use of additional vCores (this has extra cost)
* Prologic and InRiver discovery is planned during build phase of the integration solution and may impact the overall delivery plan.

## In Scope

Following interfaces are in scope

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Interface Name** | **Source** | **Target** |
| 1 | Basic Product Data (initial load) | CIMS | PIM |
| 2 | Basic Product Data (delta load) | CIMS | PIM |
| 3 | Price Data | CIMS | Demandware |
| 4 | Capture Request | CIMS | Demandware |
| 5 | Refund Request | CIMS | Demandware |
| 6 | Cancellation Request | CIMS | Demandware |
| 7 | Inventory Data | CIMS | Demandware |
| 8 | Order Status Update | CIMS | Demandware |
| 9 | Order Status Update | CIMS | SQL Cubes |
| 10 | Payment Settlement | CIMS | SQL Cubes |
| 11 | Order Creation | CIMS | JDA |
| 12 | Basic Product Data (delta load) | CIMS | JDA |
| 13 | Pre-Advice Data | CIMS | JDA |
| 14 | Order Creation | Demandware | CIMS |
| 15 | Inventory lookup | Demandware | CIMS |
| 16 | Order Creation | Demandware | SQL Cubes |
| 17 | Password Validation | Demandware | MuleSoft |
| 18 | Inventory Data | JDA | CIMS |
| 19 | Order Updates | JDA | CIMS |
| 20 | Pre-Advice Receipt | JDA | CIMS |
| 21 | Customer Returns | JDA | CIMS |
| 22 | Store Data | Intraface | Demandware |
| 23 | Customer Data | XCM | Demandware |
| 24 | Customer data Migration | Venda | Demandware |
| 25 | Customer data Migration | XCM | Demandware |
| 26 | Order History Migration | Venda | Demandware |
| 27 | Order Cancellation Email | CIMS | Cheetah Email |
| 28 | Order Shipment Email | CIMS | Cheetah Email |
| 29 | Payment Refund Email | CIMS | Cheetah Email |

## Out of Scope

Following interfaces are excluded from the original scope:

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Interface Name** | **Source** | **Target** |
| 1 | Product Category | CIMS | PIM |
| 2 | Price Data | CIMS | PIM |
| 3 | Product Category | CIMS | Demandware |
| 4 | Inventory Data on Mobile | CIMS | Mobile |
| 5 | Order Returns | CIMS | JDA |
| 6 | Order Returns | Demandware | CIMS |
| 7 | Order Cancellation | Demandware | CIMS |
| 8 | Order Cancellation | Demandware | SQL Cube |
| 9 | Order Returns | Demandware | SQL Cube |
| 10 | Payment Settlement | Demandware | SQL Cube |
| 11 | Customer Registration | Demandware | Cheetah |
| 12 | Order Creation | Demandware | Cheetah |
| 13 | Enhanced Product Data | PIM | Demandware |
| 14 | Price Data | PIM | Demandware |
| 15 | Products Data on Mobile | PIM | Mobile |
| 16 | Customer Data Lookup | XCM | Demandware |
| 17 | Email sign-up data capture | Demandware | XCM |
| 18 | Product Data | WOCP | PIM |
| 19 | Order Status Update | DTS | Demandware |

## Assumptions

* Prologic CIMS will provide a single API to perform live inventory lookup based on
  + Single SKU & single store
  + Single SKU & Multiple stores
  + Multiple SKU & multiple sores
* Prologic CIMS will export price book and inventory in XML format
* JDA will export order update and inventory data in XML format
* Mule production application will only export Demandware XML files in SFTP production server. There will be no export from Production to non-production SFTP server
* Demandware will provide a single API for payment capture, refund and cancellation
* Mule will not implement any complex business logic unless specifically stated in business requirements
* Demandware will send two different shipment ids for pre-orders and normal orders
* There will be no data masking done in Mule
* Mule will map order data between Demandware and CIMS but will not implement any logic to differentiate between US and UK orders. This will be done within CIMS based on store code, catalog code, currency code provided by Demandware
* Demandware payment capture request API will accept only single request in one service call and will return a transaction id in response. In the event of no response, Mule will resend the request. Demandware will implement any logic around eliminating duplicate requests and will not charge customer more than once for an order
* Mule will not send any shipment tracking details to Demandware
* Currency handling will not be done by Mule applications
* Order and customer export files from Demandware will not exceed 3MB size. In case Order and customer data to be exported from Demandware exceeds 3MB in Demandware, multiple files of size not more than 3MB should be exported.
* Maximum file size for files exported form JDA will be 5 MB
* Mule will not maintain state of the data.
* For customer and order migration Mule will use Venda customer Id as Demandware customer no. to ensure that Demandware is able to link orders to the customers.
* Mule will not be responsible to migrate customer password.

## Dependencies

* MuleSoft ESB environment instances are made available with required permissions
* Availability of SFTP, MS-SQL DB systems on cloud with required VPC peering to the MuleSoft CloudHub instances
* Availability of Jenkins, SonarQube, GitHub and other respective tools and application
* Demandware to share sample XML requests for each of the inbound and outbound interfaces.
* Demandware to provide two different API for payment capture request and order refund interfaces
* Providing verified and agreed detailed field mappings between various integration touchpoints listed in section [2.6](#_In_Scope).

## Outstanding Issues

* All dependencies must be resolved on time
* CIMS web service performance is critical to meet SLA for MuleSoft APIs. Any performance issues around CIMS web service needs to be resolved to meet the desired performance results for MuleSoft APIs.

## Terminology

* Middleware – MuleSoft
* Interface – Connection between 2 end points of different or same systems
* Integration – Interconnection between 2 systems
* Message – Payload
* Source - Originator of the Message
* Destination – Consumer of the message

# Technical Architecture Overview



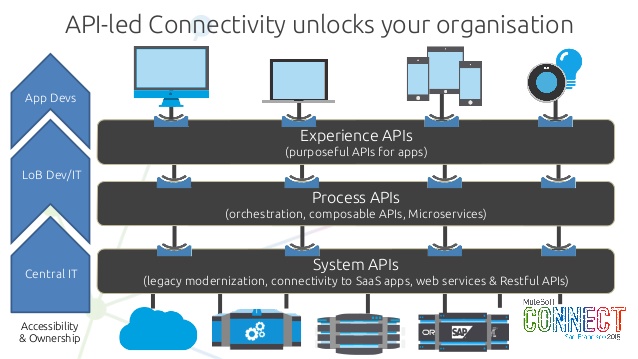


Figure . Technical Architecture Overview

It is recommended FATFACE to adopt an API-led strategy that offers several advantages over the point-to-point connection and traditional SOA based integration patterns. API-Led connectivity shifts the way IT operates and promotes decentralised access to data and capabilities whilst not compromising governance.

A framework for ordering and structuring these building blocks or APIs is also crucial. De-coupling data consumption, orchestration and connectivity concerns are critical to scalability, agility, flexibility and re-use. If data access and orchestration are decoupled, any changes to connectivity or orchestration processes are transparent to the end user. This reduces their need to understand the complexity of the systems and focus on development. API to API connectivity is simply a means of building out point to point integration. Therefore, it is recommended FATFACE adopt a multi-tier architecture that breaks-down the platform in a series of layers as described below.

#### Experience Layer

The Experience layer exposes Experience APIs, which are the means by which data can be surfaced so that its intended audience can easily consume it. The experience layer hides the complexity of IT systems from the end users, enabling them to innovate and increase productivity. It also provides a layer of security and control, providing the governance to access data that is typically controlled within IT. Experience APIs should be hosted in the Mule API Gateway and managed through the API Manager.

#### Process Layer

The underlying business processes that interact and shape this data should be strictly encapsulated independent of the source systems from which that data originates, as well as the target channels through which that data is to be delivered. Process APIs perform specific functions and provide access to non-central data and may be built by either FATFACE or Integration team. Process APIs should be hosted in the Mule ESB and can be documented in the API Manager.

#### System Layer

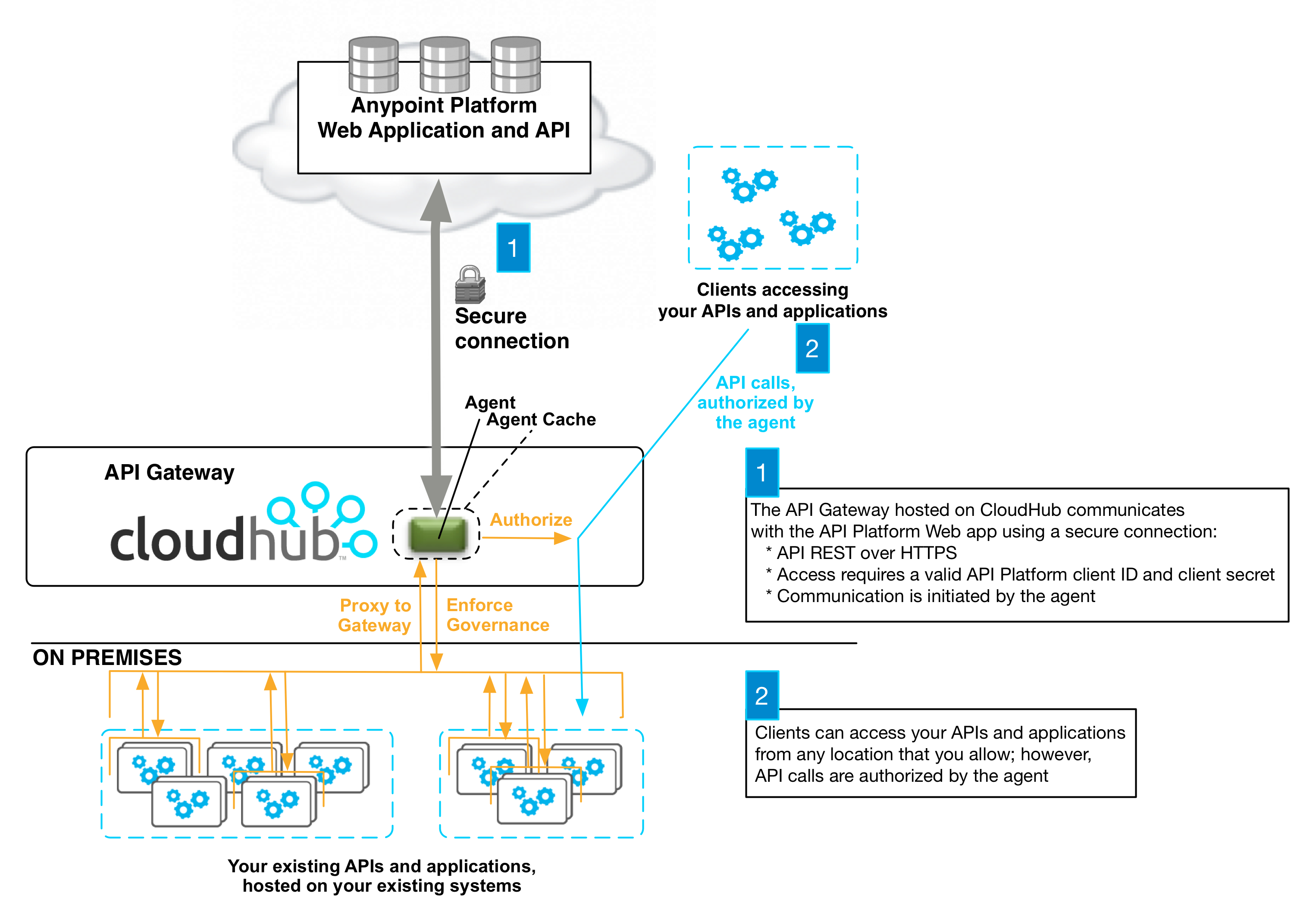
System APIs provide a means of accessing underlying systems of record and exposing that data, often in a canonical format, while providing downstream insulation from any interface changes or rationalisation of those systems. Furthermore, System APIs provide the finer grained capability that allows discreet units of functionality to be packaged in a highly re-usable fashion. System APIs will change more infrequently and will be governed by FATFACE IT given the importance of the underlying systems.

The granularity of Process and System APIs is an architectural decision (to micro-service or not) but the advantages of abstracting logic into Process and System APIs is to improve re-use, decouple security if necessary using policies, having well documented interfaces and exposing them for internal discovery quickly via the API Portal.  This is a principle of API connectivity that is borrowed from standard SOA.

## API Architecture Components

API System Architecture is the result of the configuration of three basic components:

1. Anypoint Platform for APIs: Cloud-based API registry, governance, and engagement tool
2. API Gateway: Serves as a proxy between your existing services, exposed as APIs and client applications
3. Anypoint Platform agent: Contained in the API Gateway, the agent enforces security policies created with the Anypoint Platform



The diagram above illustrates the architecture of Anypoint Platform for APIs relative to a cloud instance of the API Gateway, which proxies existing APIs, services, and applications to the Anypoint Platform. In this pairing, business logic, flows, orchestrations, execution, and message processing are proxied on CloudHub instances from wherever they are hosted. The Anypoint Platform itself contains only the service endpoint locations, service and consumer metadata, runtime policy definitions, and metrics. Thus, proxy application data remains in secure CloudHub object stores and your APIs continue to be accessible even if access to the Anypoint Platform is temporarily unavailable.

#### Agent Communication

In order to communicate with the Anypoint Platform for APIs, the agent in the API Gateway needs to authenticate with the platform. The agent and the Anypoint Platform use OAuth 2 for authentication using the client credentials workflow.

#### OAuth Authentication

When you start your API gateway (or upload an application to one of your CloudHub workers) the agent authenticates via OAuth with the Anypoint Platform API, which checks that the agent's client credentials are valid. The agent then sends a list of all endpoints configured on the API gateway to the Anypoint Platform API. Any matching endpoints registered in the Anypoint Platform are now tracked by your agent.

#### Security and Reliability

The Anypoint Platform for APIs is a fully multitenant application running on top of Amazon Web Services (AWS). The Anypoint Platform runs inside a VPC (cloud VPN). Data, metrics, and metadata cannot be accessed across organizations.

Although the Anypoint Platform can manage and enforce the runtime security of services, the services themselves remain wherever they are deployed. Only the configuration of the policies and the metadata about the services is stored in Anypoint Platform; the actual enforcement of the policies and contracts occurs "on location" at the gateway.

All communication between the Anypoint Platform and the agent, whether that agent on premises or on CloudHub, is performed through REST calls; OAuth is used for agent authentication. This counters any attempt to compromise the infrastructure through man-in-the-middle attacks or spoofing of services. The Anypoint Platform agent initiates communication with the Anypoint Platform API, also preventing any type of DNS type attack on your infrastructure.

The runtime components are insulated from external network outages since they store a local cache and can continue to operate even if the Anypoint Platform were to become unavailable. Regardless, MuleSoft maintains an SLA of 99.99% for Anypoint Platform for APIs.

## FATFACE Architecture

FATFACE recognizes the advantages offered by the three-layer architecture. However, due to time and cost constraints FATFACE wants the first phase of the integration to be batch integration that reduces the point-to-point integration between their systems and helps migrating their current web platform from Venda to Demandware before their peak sale period. The main factor contributing to this decision is

* Demandware currently does not offer API interfaces for integration and their preferred approach to integration is import-export of XML files via SFTP server. In order to consume API exposed from Mule, Demandware will need to do custom development that is currently out of scope for the e-commerce project.
* The legacy backend systems heavily depend on the file import and export. Customisation will be required in the backend systems to allow synchronous communication. If there are any business process changes required in the legacy backend systems, it may also need additional training to manage the systems.

Considering above requirements and constraints, following architecture has been proposed for the first phase of the integration project. However, FATFACE has shown interests in migrating some of these batch interfaces into APIs in the later phases of the projects.

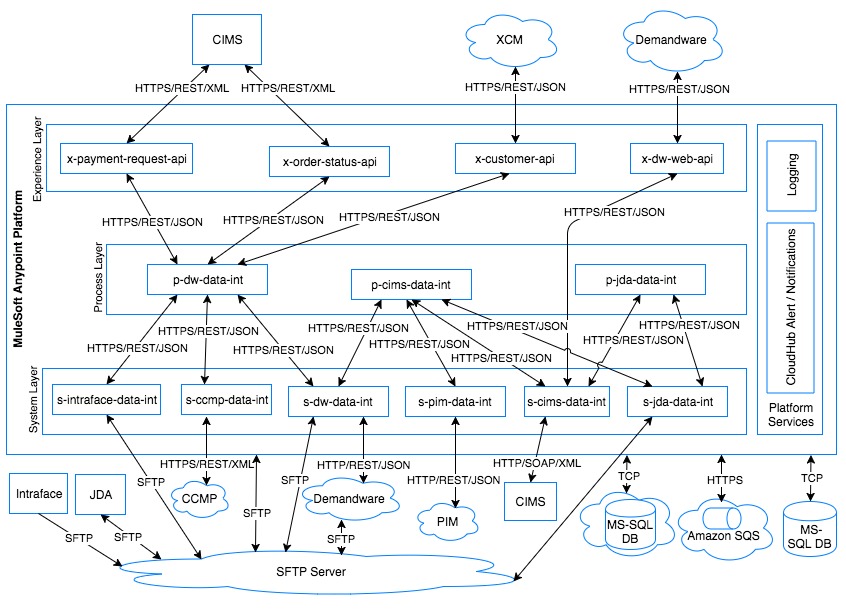


Figure . FATFACE Architecture

* The applications in experience layer will expose RESTful API for Demandware, XCM and CIMS.
* Behind the scenes, the experience APIs will make use of the process APIs and/or system APIs to perform their specific functions.
* If any data processing or orchestration of API calls needed, a process API will be developed and experience API will invoke process APIs to fulfil the business requirements
* The Process layer will currently host all batch applications where Mule will be responsible to pull request data from third party web services and send the data to the target system via SFTP server or third party web services call after the required processing and message translation.
* The process layer will also contain process APIs that can except request from experience layer or system batch processes.
* Applications deployed in system layer will be responsible to integrate with various backend systems. These application will either expose Restful APIs or receive messages via a JMS queue for asynchronous processing of data.
* The system layer will host applications that polls for a file from the SFTP server, transforms the data and invokes process API to process the data to the expected target system.
* If applications in process layer receives data in batch and target system expects data one at a time, Mule will publish data to a JMS queue so that application in the system layer can receive data from the queue and send to the target system one at a time.
* Common logging and auditing framework component will be used in each of these applications for standard logging of data and auditing of incoming and outgoing messages wherever required.

It is recommended FATFACE to later migrate the applications into process layer from batch to restful APIs and develop experience APIs for Demandware to consume.

## Components of Application

## ESB Components

MuleSoft provides a whole lot of connectors and components out of the box. Listed below are the components to be used in this solution.

|  |  |
| --- | --- |
| **Component** | **Description** |
| HTTP Connector | Mule supports HTTP and HTTPS Request and Response over HTTP connector to accept the requests and respond to the consumer over HTTP/HTTPS protocol. |
| Poll | Used to poll a resource based on the given time schedule. If the application is deployed on Cloud-hub with certain frequency, users can run interface immediately from Scheduler > Select respective poll schedule > run now. |
| SFTP | The SFTP Connector implements a file transport channel so that your Mule application can exchange files with an external SFTP server. You can configure SFTP as an inbound endpoint (which receives files) or outbound endpoint (which writes files to the SFTP server). |
| Amazon SQS Connector | The Amazon SQS connector allows to send and receive messages to and from Amazon SQS. |
| APIKitRouter | This component is used to route the API calls to the specific endpoint from the common API interface. |
| Alerts and Logging | Logger Components will be used for debugging and for error logging. The Mule Logger component is based on log4j2. Custom logging framework developed by WHISHWORKS will also be used across the application for logging. |
| DataTransformers | These are Mule out of the box transformers as well as custom APIs which will be used for data transformation and enriching (example: Object to String, ByteArraytoString) |
| Error Handlers | Default and Custom error handlers to raise default/custom exceptions. |
| DataWeave | DataWeave is MuleSoft built in Transformer which is used for mapping, transformation and integration process (example json to xml), from a given input to the given output. |
| Java Custom Components | Java custom components will be used as message processors for functionality like part code mapping, static data mapping, removing unwanted characters from data etc. |
| Configuration Data | These are list of configuration files used, which contain properties like DB, ECR 2.0 credentials, domain configurations, database connection details, external system connection details etc. |
| Configuration Loader | Configuration loader is used to load the configuration files and cache them so that the properties can be read by Java components. |
| Database Connector | ESB Adaptor used to connect to the database for maintaining transactional data.  This adaptor uses Connection pooling to the underlying database and also supports defining SQL queries and for batch inserts/updates for Audit logs |
| Basic authentication | Mule Server will authenticate the Trade partners using BASIC authentication over HTTPS. |

## Database Components

Preferably a MS-SQL Database on AWS Cloud will be used for the current CloudHub applications. There will be two schemas created in the database as detailed below:

* MULE-DB - Integration Data Base schema that contains tables required by common framework components such as audit logs and lookup data.
* FATFACE-DB – contains business rules required by application such as order to customer mapping for initial load of customer and order history from Venda and XCM.

## Other Components

WHISHWORKS will build a common logging framework that provides the functionality of logging and auditing the Mule message. Its purpose is to keep a track of every request being made and the response being generated. It is an independent component that could be used across multiple applications.

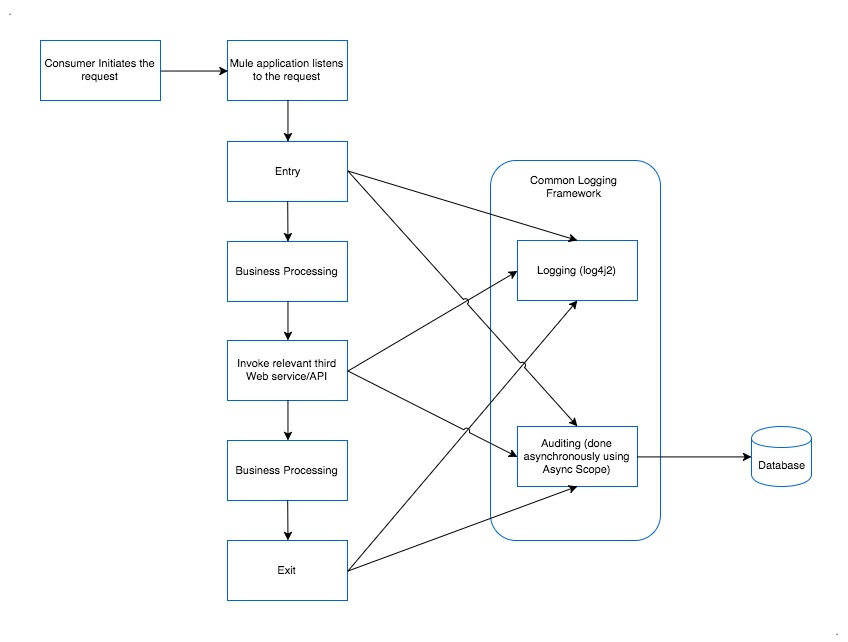


Figure . Common Framework

**Functionality**

* Logging and auditing will be done at critical points during Mule message lifetime
* Entry
* Exit
* Invoking third party request
* Response from the external systems
* Logger uses log4j2.xml configuration for logging message asynchronously.
* Logs will have information of
* Logging position (entry-logger, exit-logger, call-entry-logger, call-exit-logger, process-logger, warn-logger-flow, error-logger-flow).
* Application Name
* Service Name
* Flow Name
* Request Id
* Message Id
* Correlation Id
* Message
* Auditing is done asynchronously using async-scope in Mule flows.
* Auditing information will contain:
* Flow Status (describing the status of flow)
* Record create time
* Message
* Auditing functionality could be switched ON/OFF based on a configuration at application level.
* The framework component requires a database for storing the audit logs. MS-SQL database server on the Amazon RDS server will be used for Fatface audit.

The common framework development is expected to be done in FATFACE development environment where the external partners like WHISHWORKS can develop and test the interfaces. If this arrangement is not available, then WHISHWORKS uses their development environment to develop and test these components.

# Application Design



## Architecture Views

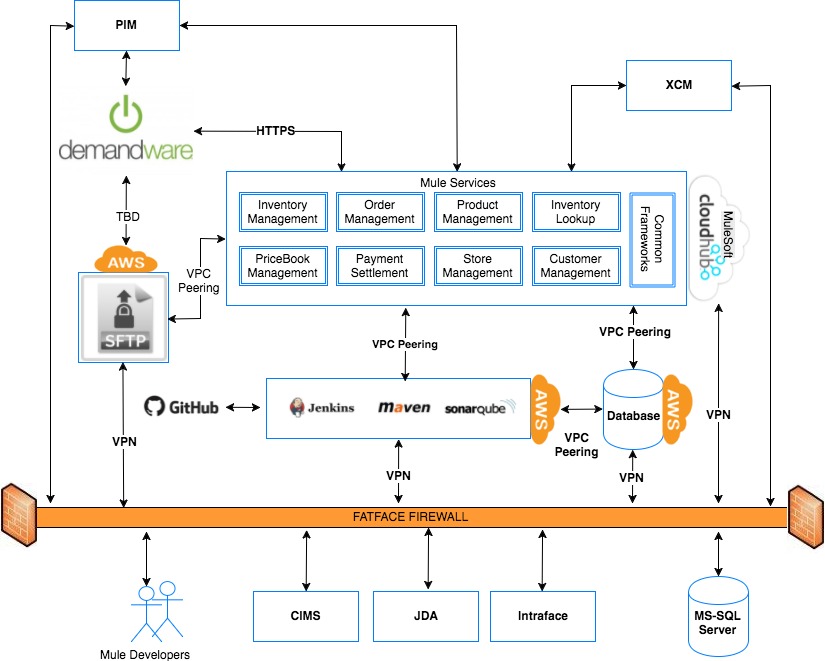


Figure . Architecture View

Above diagram shows a high level view of the proposed architecture.

* An integration database will be installed in AWS RDS instance.
* GitHub will be used as the code repository and AWS EC2 server hosting CI tools will have access to GitHub to build applications.
* CI server will also have access to MuleSoft CloudHub server to deploy Mule applications on the Mule runtime
* MuleSoft CloudHub will have access to database hosted in AWS RDS instance for storing audit and other mapping data.
* CloudHub will have access to SFTP server hosted in a separate AWS EC2 server.
* CloudHub will have access to FATFACE on premise systems
* The SFTP server hosted in AWS EC2 instance will also be accessible from the on premise systems

The Virtual Private Cloud (VPC) offering allows to virtually create a private and isolated network in the cloud to host the [CloudHub workers](https://docs.mulesoft.com/runtime-manager/cloudhub-architecture#cloudhub-workers).

The design here connects MuleSoft CloudHub to FATFACE network via a secure VPN connection. This allows CloudHub workers to access resources behind the corporate firewall. The design here leverages an IPsec gateway for VPN connectivity.

Connecting a network to a CloudHub VPC with an IPsec VPN connection is shown in the diagram below:

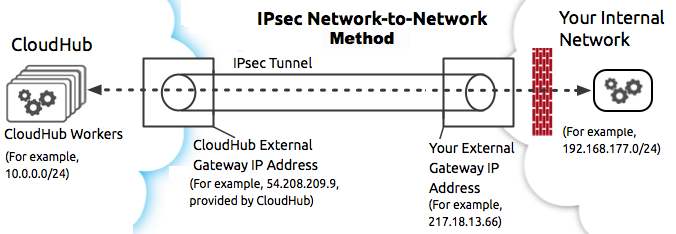


Figure . IPsec VPN connection

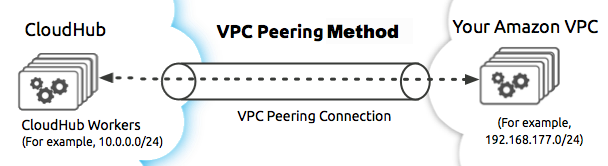
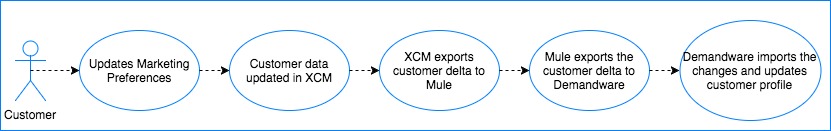
The services hosted on AWS can connected by peering CloudHub VPC and AWS VPC.  
The diagram below illustrates connecting a CloudHub VPC and Amazon VPC together directly through VPC peering:  
[](https://docs.mulesoft.com/runtime-manager/_images/CHVPC05.png)

Figure . VPC Peering connection

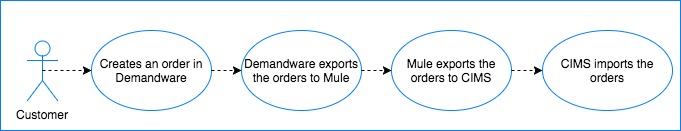
## Use Case View

Following pictures give a high level view of business use cases that requires integration via Mule. The pictures are self-explanatory. Detailed view will be part of the low level design for each of these use cases.

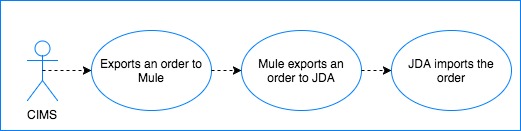
## Update Marketing Preferences



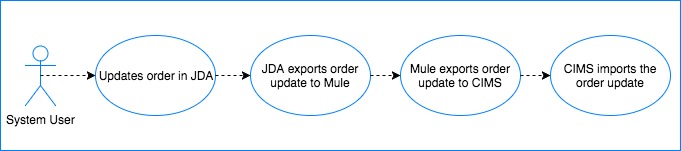
## Create Order in Demandware



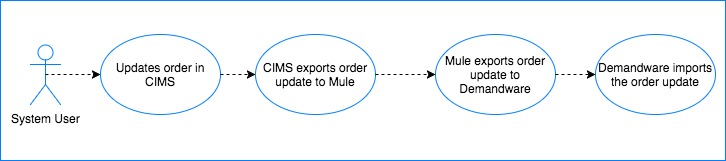
## Export Order to JDA



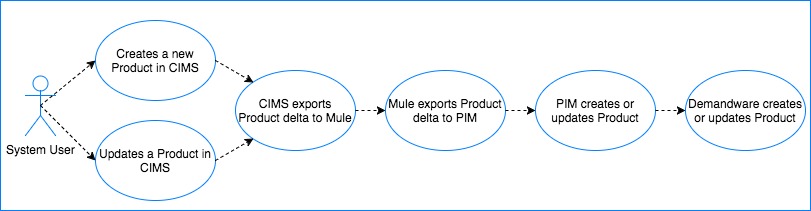
## Order Update in JDA



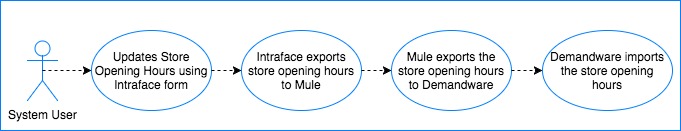
## Order Update in CIMS



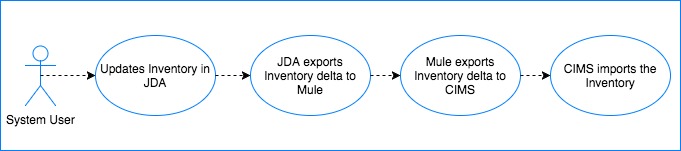
## Product Update in CIMS



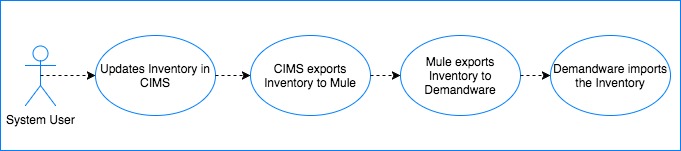
## Store Opening Hours Update



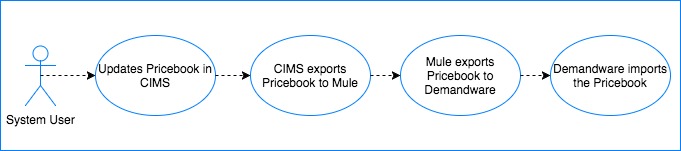
## Inventory Update in JDA



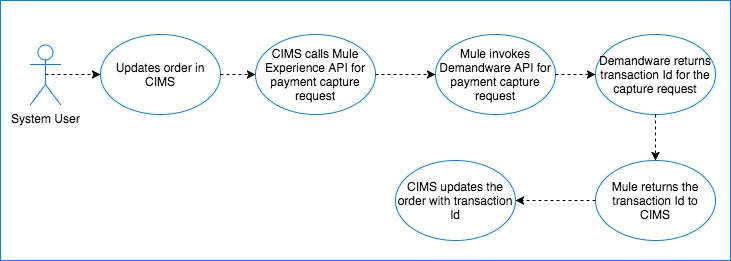
## Inventory Update in CIMS



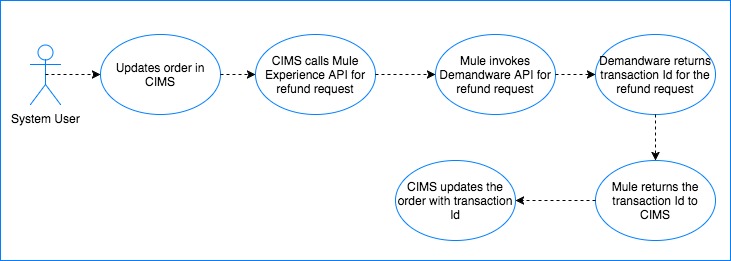
## Pricebook Update in CIMS



## Payment Capture Request



## Refund Request



## Logical View

Mule applications will be broken into 3 logical layers as shown in the following picture:

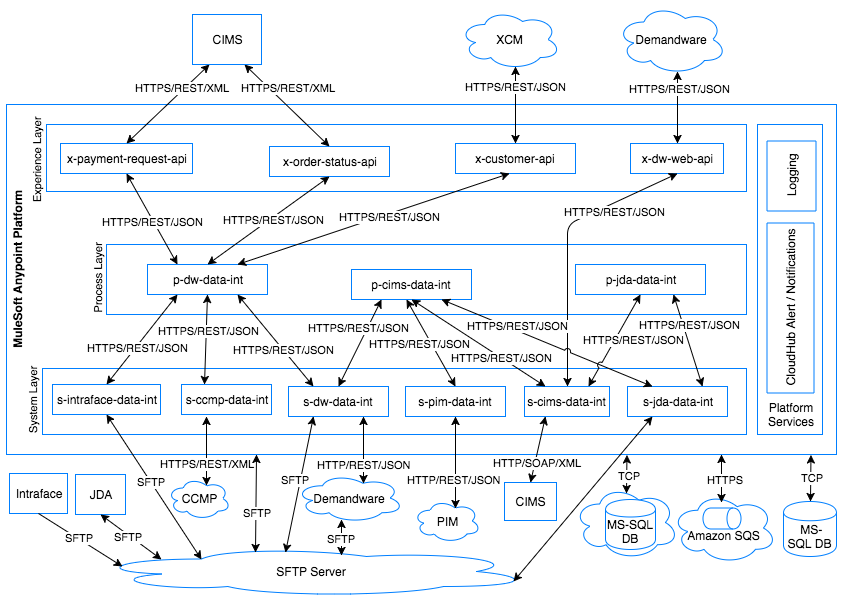


Figure . Logical View

1. Experience Layer: This layer will host following applications:
   1. x-dw-web-api
   2. x-payment-request-api
   3. x-order-status-api
   4. x-customer-api
2. Process Layer: This will host following applications:
   1. p-cims-data-int
   2. p-jda-data-int
   3. p-dw-data-int
3. System Layer: This will host the following applications:
   1. s-pim-data-int
   2. s-jda-data-int
   3. s-ccmp-data-int
   4. s-dw-data-int
   5. s-cims-data-int
   6. s-intraface-data-int

Each of these applications will have one or more APIs/Flows containing one or more operations.

Applications in experience layer will never integrate with any target system directly rather integrate to the target system via a process API or system API.

System APIs will provide an abstraction for each systems and will be developed in such a way that those can be used by one or more process or experience APIs. Applications in this layer will also interface with SFTP server to poll for files, transform data and send data and invoke process API to send data down to the target systems.

Applications in process layer will provide integration between experience APIs and system APIs where it will handle all the complexity and execute any business rules. Application in this layer will also handle any batch requirements that FatFace has.

For more details on the individual APIs within above applications, please refer [Mule application deployment in CloudHub](#MuleApplicationDeploymentInCloudHub) in section [4.1.5](#_Deployment_View_1).

## Implementation View

##### Experience API Implementation

All experience APIs will follow one of the following patterns

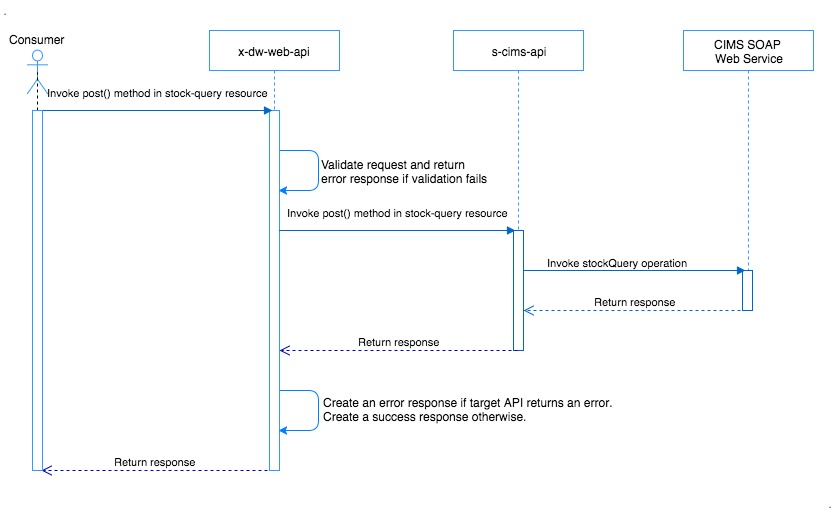


Figure . X-Layer: Pattern-1: X-API calling S-API

As shown in the above picture, some experience API will directly invoke the system API. This pattern will be used if system API is a wrapper API providing abstraction on top of the target system and implementation logic does not need any orchestration of calls or complex business logic execution to be done in a process layer.

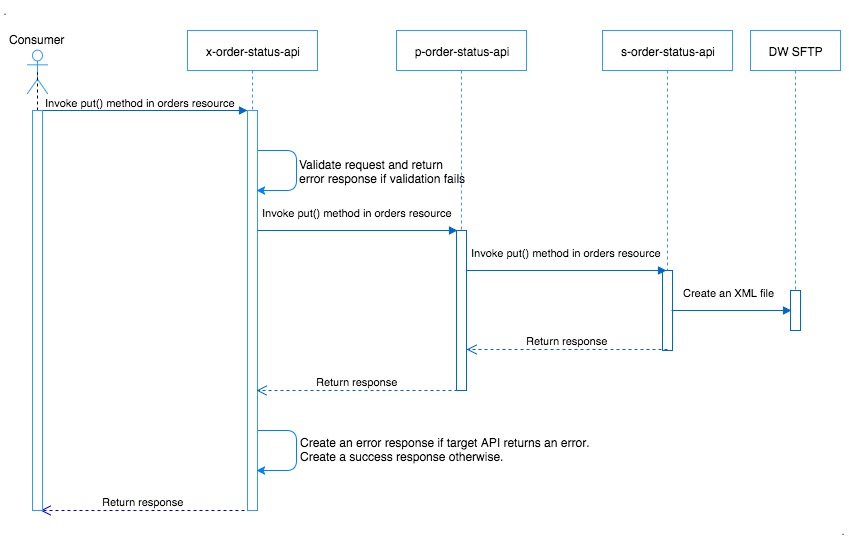


Figure . X-Layer: Pattern-2: X-API calling P-API and P-API calling S-API

As shown in the above diagram, some experience APIs will use 3-layer API architecture. The X-API will be used to expose an API specific to a client (CIMS in this use case) requirements. Process layer API is created to be a reusable API which can accept message from the X-API and can also receive message from other system batch process (DTS order status in this use case). The process layer API further calls the system API which finally sends the required data to the target system (SFTP service in this case).

##### Process API Implementation

Any interface that accepts the data in batch and needs to send the data to the target system one at a time will follow the pattern as shown below for order management process API.

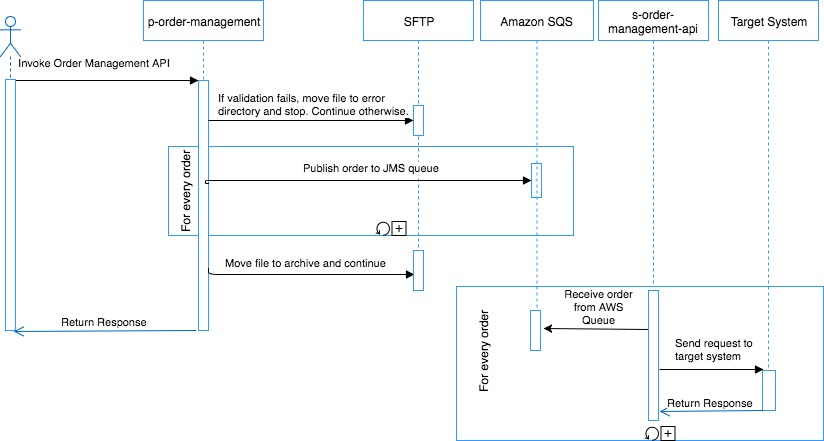


Figure . P-Layer: Pattern-1: P-API batch request to S-API single request

The inventory data processing sequence diagrams depicts the implementation pattern to be used for all interfaces where source and target systems both expect single or batch data.

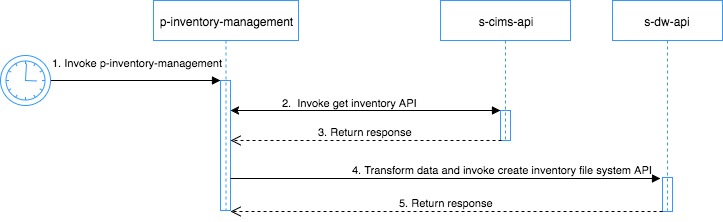


Figure . P-Layer: Pattern-3: Data pulled by calling a 3rd party web service

##### System API Implementation

Following sequence diagrams depict various implementation patterns to be used for the system API.

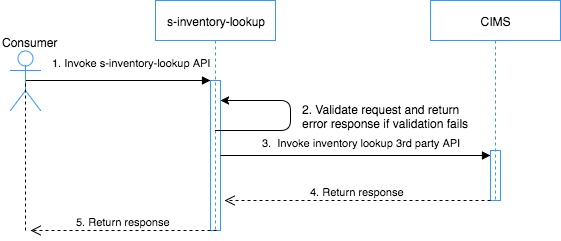


Figure . S-Layer: Pattern-1: Pulling data from 3rd party web service

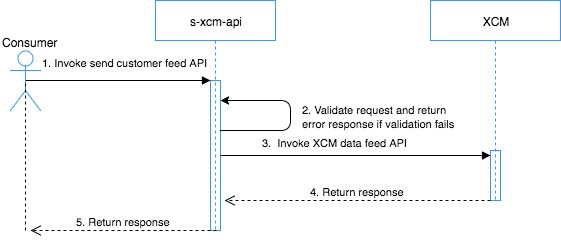


Figure . S-Layer: Pattern-2: Pushing data by calling a 3rd party web service

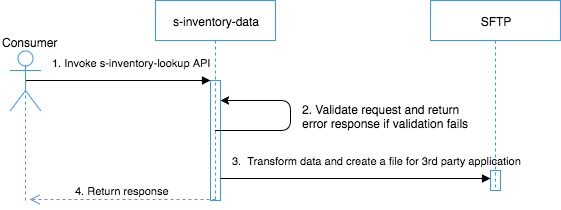


Figure . S-Layer: Pattern-3: Pushing data on SFTP server

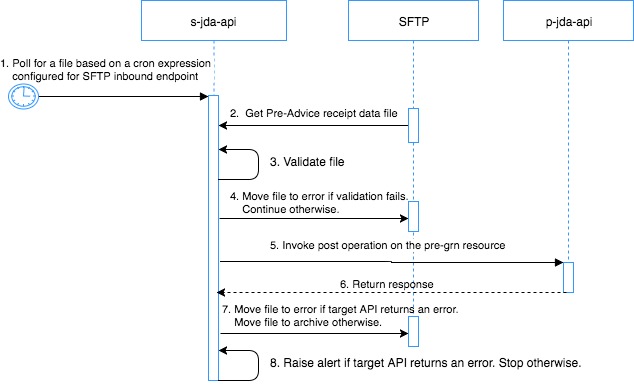
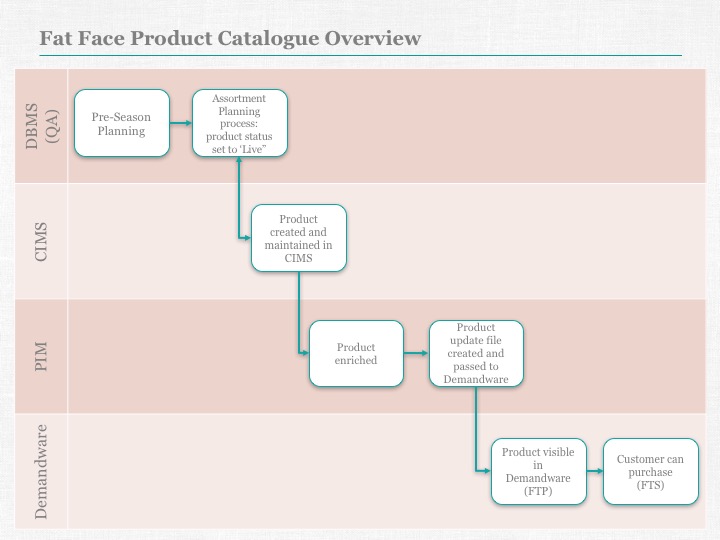


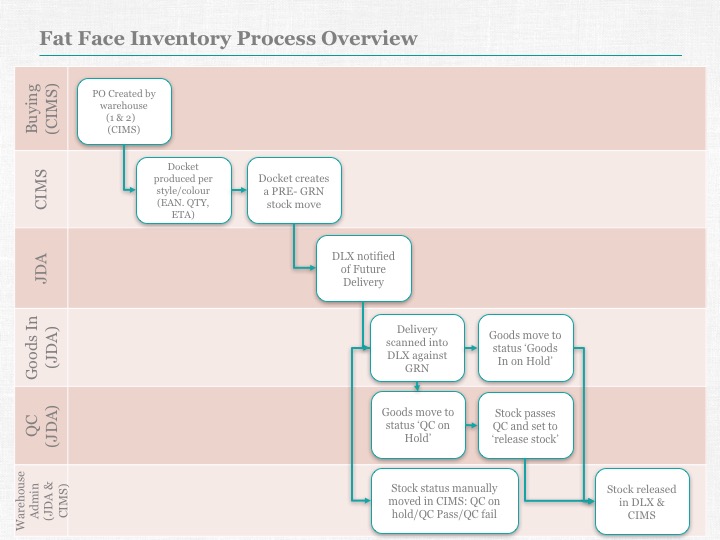
Figure . S-Layer: Pattern-4: Pulling data from SFTP server

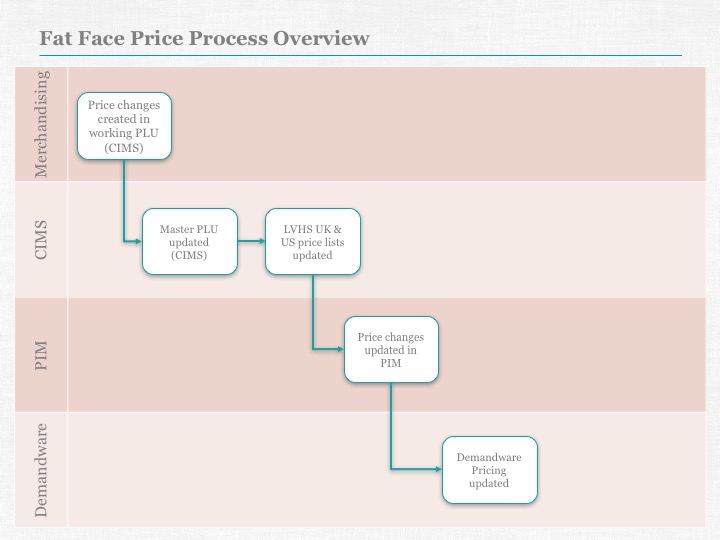
## Process View

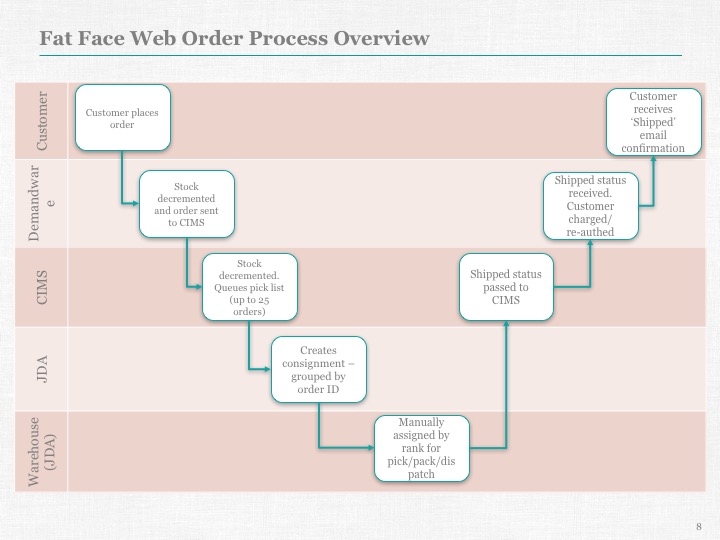
Following diagrams gives a high level view of the processes involved in product catalog, inventory, pricing and order.

Note: Actual process view for the new interface is being worked on by Gary Bolton and John Ray. Below diagram may change based on any changes in the business processes.









## Deployment View

**Infrastructure Setup**

Following diagrams depicts the infrastructure deployment view.

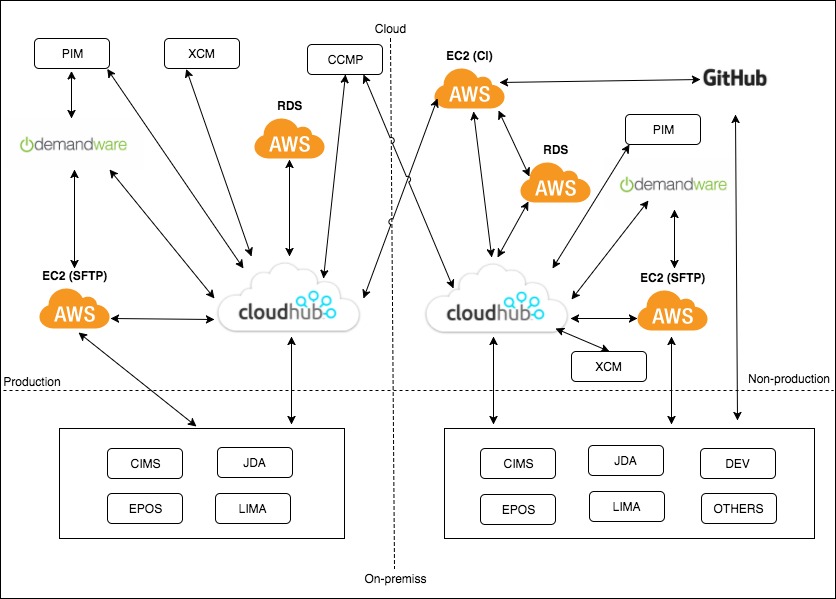
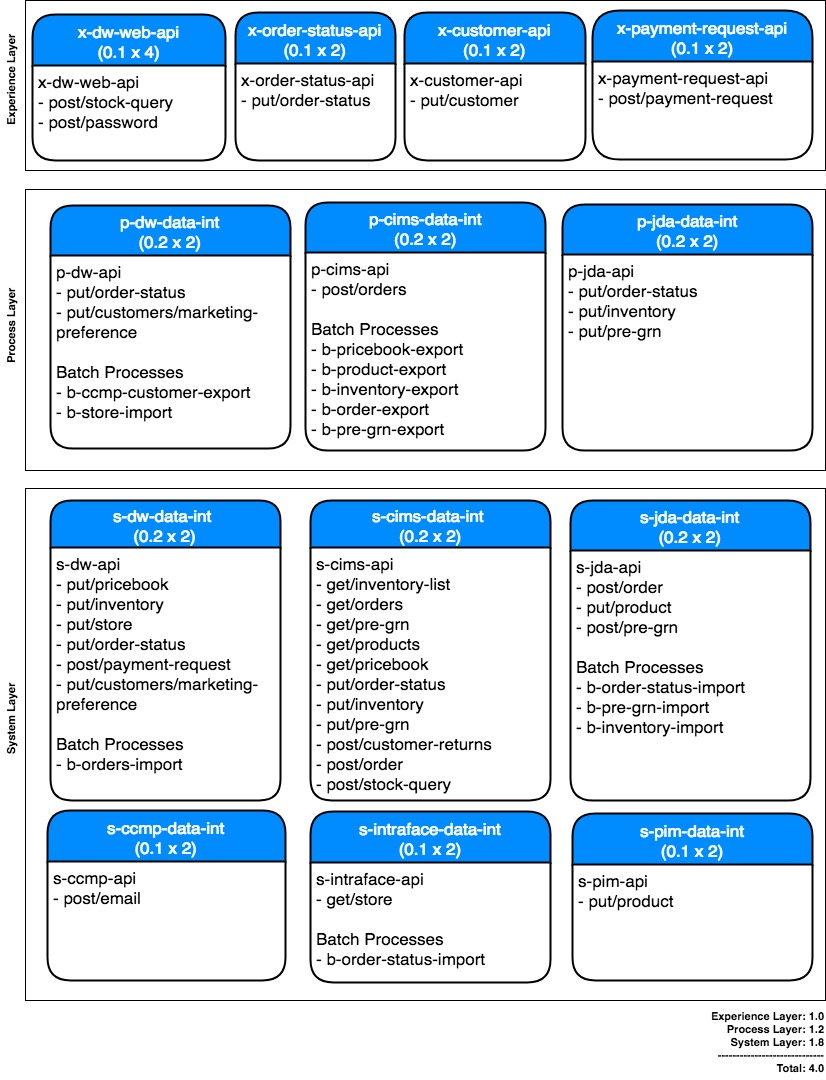


Figure . Deployment View

* MuleSoft runtime and API Gateway will be deployed on CloudHub
* Amazon RDS server will be used to hold audit and staging data for Mule application
* Amazon EC2 server will be used to install CI related tools (Jenkins, Maven & SonarQube)
* Amazon EC2 server will be used to install Serv-U MFT SFTP Server
* GitHub on cloud will be used for source code repository
* Both MuleSoft production and sandbox environments will have access to CCMP production environment as CCMP does not have any test/sandbox instance.
* The CI server will have access to both sandbox and production MuleSoft environments so that deployment can be automated, if required.

**Mule application deployment in CloudHub**

Following table gives a high level view of Mule application deployment in CloudHub. Note that the API and application names are logical names and may change as per the best practices when creating a low level design for the same. The text in the brackets (e.g., 0.1 X 2) in the headers if the individual boxes represent the MuleSoft worker size and no. of workers assigned to the application respectively. For more details on MuleSoft workers, please refer MuleSoft documentation [here](https://docs.mulesoft.com/runtime-manager/cloudhub-architecture#cloudhub-workers).



Above table shows the mule application deployment view for Production. Applications in Preproduction/UAT environment may be deployed using a single worker. Dev and SIT environment has only 1 vCore each hence above deployment view is not applicable for Dev and SIT. Dev and SIT may use 0.1 vCores per application or 0.2 vCores per application being tested. Any performance and load testing shall be carried out in UAT environment only.

**API Endpoints**

Based on the best practices the naming convention for each API in various environments will be as below.

1. For Development environment, the word ‘dev-’ is prefixed to the app name.

Example: dev-x-customer-api.eu.cloudhub.io

1. For SIT environment, the word ‘test-’ is prefixed to the app name.

Example: test- x-customer-api.eu.cloudhub.io

1. For UAT, the word ‘preprod-’ is prefixed to the app name.

Example: preprod- x-customer-api.eu.cloudhub.io

1. For Production environment, no change to the app name.

Example: x-customer-api.eu.cloudhub.io

Note: As most of the Fatface backend systems have only two environments (test and production), Fatface may decide to do their initial integration testing using either MuleSoft dev or sit environment. Once initial testing is done, business testing and performance testing shall be done in UAT environment.

## Design Considerations

Following design considerations are for all the implementation across this HLD.

1. APIs exposed by MuleSoft will be stateless, hence no state or functional data is maintained on the ESB.
2. Mule will expose REST/JSON experience API for password validation, stock query and customer APIs. REST/XML will be used for payment settlement and order status update as consuming application (CIMS) supports XML over JSON.
3. Mule experience API will be a secure Restful web service that will use HTTPS and client-id security enforcement.
4. All experience APIs will use port 8082 which will by default use CloudHub load balancer and will get exposed over port 443.
5. There is no specific need of securing system APIs as those will not be directly exposed over internet. However, Whishworks recommends to keep all layers of API secured using client-id security enforcement and using HTTPS across all layers.
6. All process and system APIs should use port 8092. Load balancing will be achieved using DNS round robin supported by MuleSoft HTTP requester component.
7. WHISHWORKS recommends to use API led connectivity approach for all integration requirements. FATFACE has opted batch integration for phase-1 of the e-commerce integration project due to time and cost constraints. FATFACE shall try to adopt API led connectivity approach in the later phases of the projects. Following disadvantages associated with the batch integration solution have been highlighted to FATFACE:
   1. Makes the communication asynchronous
   2. Adds an additional layer of failure point
   3. Additional infrastructure to manage and maintain the SFTP Server
   4. Handling Errors and exceptional scenarios is not easy
   5. Housekeeping activities on file storage vs processed files vs failed files etc.
   6. Scalability / Extendibility - The Solution is not easy to either scale or extend to share the same data with future consumers (Partners, ePOS application etc.)
8. Components, modules, and functions will define a contract or interface specification that describes their usage and behavior clearly. The contract will describe how consumers can access the internal functionality of the component, module, or function. API documentation and contract information will be published to the consuming partners through API portal. A service design documents (SDD) will be created for each API which cover low level API design and implementation details.
9. The development methodology will be based on Maven build tool, with MUnit test cases written for Mule flows and Junit test cases for each java code. The test cases will be run on every build, ensuring minimal regression errors. The test reports will be published for each build in continuous integration system (Jenkins).
10. The current consolidation of Mule application and allocation of vCores are done for low to normal data processing. If the data size or volume increases, application scaling may be needed based on memory or throughput requirements. This may require buying additional vCore licenses for CloudHub.

## Design Solutions

Following file structure is proposed to be created on SFTP for MuleSoft ESB to carry out file processing & archival for various interfaces.

ROOT [FatFace/sftp/MuleSoftIntegration/<environment>]

|  |  |  |
| --- | --- | --- |
| **Directory** | **Subdirectory** | **Description** |
| /error | /orderData/jdaOrderUpdate | Order update data file received from JDA |
|  | /inventoryData/jdaInventory | Inventory data file received from JDA |
|  | /storeData/intraStoreOpeningHours | Store opening hours data file received from Intraface |
|  | /grnData/jdaPreGRN | PreGRN file received from JDA |
|  | /orderData/dwOrder | Order data file received from Demandware |
|  | /orderData/intraOrderUpdate | Order update data file from DTS/Intraface |
| /archive | /incoming/orderData/jdaOrderUpdate | Order update data file received from JDA |
|  | /incoming/inventoryData/jdaInventory | Inventory data file received from JDA |
|  | /incoming/storeData/intraStoreOpeningHours | Store opening hours data file received from Intraface |
|  | /incoming/orderData/intraOrderUpdate | Customer data file received from Demandware |
|  | /incoming/orderData/dwOrder | Order data file received from Demandware |
|  | /incoming/grnData/jdaPreGRN | Pre-GRN data file received from JDA |
|  | /outgoing/orderData/jdaOrder | Order data file exported to JDA |
|  | /outgoing/storeData/dwStoreOpeningHours | Store opening hours data file exported to Demandware |
|  | /outgoing/productData/jdaProduct | Product data file exported to JDA |
|  | /outgoing/inventoryData/dwInventory | Inventory update file exported to DW |
|  | /outgoing/orderData/dwOrderUpdate | Order update file exported to DW |
|  | /outgoing/priceData/dwPriceUpdate | Pricebook file exported to DW |
|  | /outgoing/grnData/jdaPreGRN | Pre-GRN data file received from JDA |

Above directories need to be created for each Mule environment. Mule will be polling for files exported from Fatface systems such as Intraface, JDA & Demandware from their respective SFTP server directories. Based on the processing result, Mule will move the files to its respective error or archive directories. If Mule needs to export a file to a third party SFTP server location, Mule will create a file on its own SFTP outgoing directory and copy it to its respective 3rd party SFTP server location.

Examples:

1. Store Opening Hours Update
   1. Mule receives a file at “Intraface/StoreOpeningHours” and moves it to “FatFace/sftp/MuleSoftintegration/dev/archieve/incoming/storeData/intraStoreOpeningHours”.
   2. If file is parsed successfully, Mule creates a file in the Demandware inbound directory “SalesforceCC/toCC/stores” and also copies the file to “FatFace/sftp/MuleSoftintegration/<environment>]/archive/outgoing/storeData/dwStoreOpeningHours” directory.
   3. If there is an error in processing the file, the incoming file will be moved from “FatFace/sftp/MuleSoftintegration/dev/archieve/incoming/storeData/intraStoreOpeningHours” to “FatFace/sftp/MuleSoftintegration/<environment>]/error/storeData/intraStoreOpeningHours” directory.

**JDA SFTP directory structure**

|  |  |  |
| --- | --- | --- |
| **Directory** | **Subdirectory** | **Description** |
| /Inbound | /Order | JDA to import orders |
|  | /PreAdvice | JDA to import Pre-Advices/Pre-GRNs |
|  | /Product | JDA to import Products/SKUs |
| /Outbound | /PreAdviceReceipt | JDA to export Pre-Advice/Pre-GRN receipt. |
|  | /OrderStatus | JDA to export order status changes |
|  | /Inventory | JDA to export Inventory status changes and Customer Order Returns |

**Intraface SFTP directory structure**

|  |  |  |
| --- | --- | --- |
| **Directory** | **Subdirectory** | **Description** |
| /StoreOpeningHours |  | Intraface to export store opening hours PSV file |
| /DeliverToStore |  | JDA to export DTS order status PSV file |
| /VendaCustomer |  | Venda to export customer migration files |

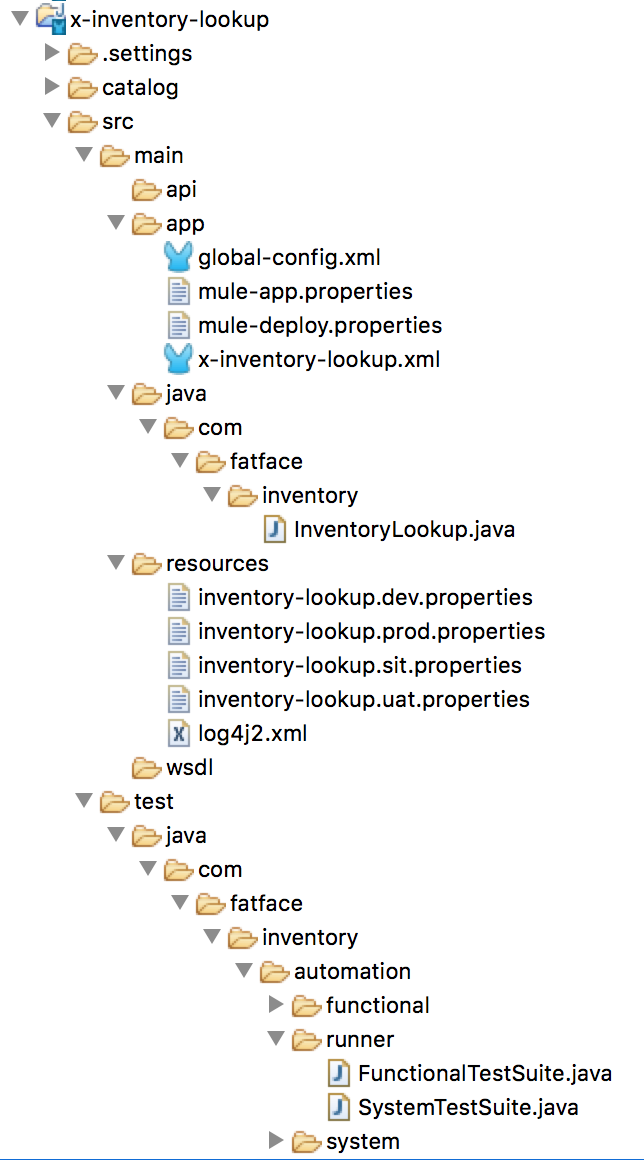
**Demandware SFTP directory structure**

|  |  |  |
| --- | --- | --- |
| **Directory** | **Subdirectory** | **Description** |
| /migrationToCC | /orders | Order migration files to import in Demandware |
|  | /customers | Customer migration files to import in Demandware |
| /toMule | /orders | Orders exported from Demandware for Mule API to process |
| /toCC | /pricebooks | Pricebook file to import in Demandware |
|  | /inventory-staging | Inventory file to import in Demandware staging environment |
|  | /inventory-production | Inventory file to import in Demandware production environment |
|  | /orders-updates | Order status update file to import in Demandware |
|  | /stores | Store opening hours file to import in Demandware |
|  | /customers | Customer marketing preference file to import in Demandware |

Note: The reason to maintain a copy of the outgoing file form MuleSoft applications on the MuleSoft SFTP server was to ensure that a copy of the file is kept as backup on the SFTP server so that it can be used later to manually copy the file on the target 3rd party SFTP server if attempt to copy the file from Mule application fails. Fatface has decided to use a single SFTP server on AWS EC2 instance which will be shared by all Fatface applications such as JDA, MuleSoft, Demandware and Intraface. Due to this it’s not essential for Mule application to maintain a copy of the outgoing file.

**The Mule application Project Structure**

Following is a sample project structure to be used for x-inventory-lookup Mule application:



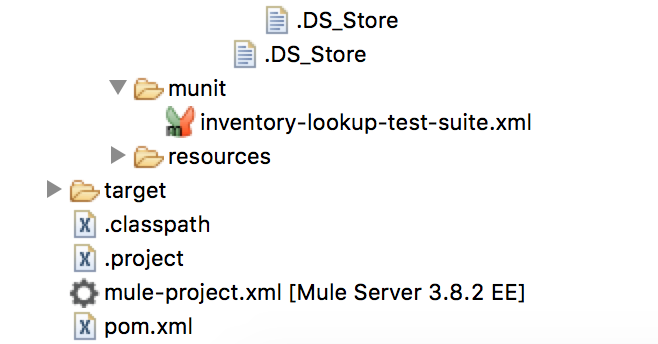


Figure . Mule API Project structure

1. All global components will be defined inside global-config.xml
2. x-inventory-lookup.xml file define the experience API flow that will internally invoke s-inventory-lookup API to fetch the live inventory data
3. Properties file will be catagorised as environment specific properties files and application specific properties files and all will be kept inside the project “src/main/resources” directory
4. Environment specific properties files will have <filename>.<env>.properties. as show in [figure 10](#Fig10).
5. Any passwords and key properties will be encrypted using AES algorithm
6. Munit tests will be written for APIs and kept in “src/test/munit” directory

For best practices around the naming convention of the mule components, refer the best practices guide links given in [reference](#Ref) section of this document.

Applications in all Mule layers will follow the same project structure.

**Development Setup**

1. Version control and source code management will be carried out using GitHub. Code will be maintained in a central repository location on cloud via GitHub.
2. Off-shore and On-site team will share the code, stored in GitHub using the unique registered email id.
3. Maven will be used for build and manage process.
4. Latest Java SE Runtime environment 8 will be used.
5. A MS-SQL database will be required for storing audit details. The database will be installed on AWS RDS server.
6. SoapUI NG Pro will be used for testing any API deployed on CloudHub.
7. Jenkins will be used as Continuous Integration tool and will be installed on AWS EC2 server.
8. Deployment of respective Mule applications (.zip) to CloudHub will be done via ARM (Anypoint Runtime Manager)
9. Communication and Data exchange between Demandware and Mule will happen via predefined directories in the shared SFTP server.
10. Poll component having cron expression will be used to trigger any batch process in Mule.
11. SFTP inbound endpoint will be used to read files from SFTP server based on a cron expression maintained in a property file.
12. [Figure 18](#Fig9) depicts overall environment, from network architecture point of view. Non-production section is for Dev, SIT and UAT and production section is for live environment only.

## Mule Connectors

1. FTP/SFTP Connector
2. Database Connector
3. VM Connector
4. REST Web Service Connectors / HTTP Request Connectors
5. Mule Requester Module
6. HTTP/HTTPS Listener
7. Web Service Consumer
8. Amazon SQS Connector

## Mule Message Processors

Following message processors will be used:

1. APIKitRouter: To route the requests to internal sub flow with in Mule applications.
2. Logger: To log the messages to the system logs.
3. Transformers: DataWeave will be used to transform the messages from XML to JSON or JSON to XML.
4. Filters: Schema validator filters will be used for validating the messages in incoming requests.
5. Routers: Choice, Splitter, Aggregation processors will be used.

## Data Model

1. A Canonical data model approach will be taken for exchanging data/records between different systems.
2. XML in case of Batch process and JSON in case of service integration.

## Protocols and Formats

1. REST/JSON will be used over HTTPS to expose stock query, password validation and customer experience APIs.
2. REST/XML will be used over HTTPS to expose payment settlement and order status experience APIs.
3. All process and system APIs will use REST/JSON over HTTPS to privately expose an endpoint which can be accessed within Fatface VPC.
4. XML will be used as the preferred format for any file based batch integrations. However, some interfaces may use PSV files.

## NFR Solutions

MuleSoft API specific NFR is captured in the NFR spreadsheet given at the below link.

[Fatface NFR](https://fatface.atlassian.net/wiki/download/attachments/92322210/Fat%20Face%20-%20NFR.xlsx?api=v2)

Below sections provides other Non-Functional Requirement for FATFACE Demandware integration project.

## Availability

To provide high availability, scalability (horizontal) and redundancy, an application should be deployed to multiple workers in production. MuleSoft automatically distributed multiple workers for the same application across two or more data centers (Amazon Availability Zones) for maximum reliability. MuleSoft ESB Runtime Workers currently offer an SLA of 99.99% uptime (excluding scheduled upgrades).

## Scalability

Currently Production environment consists of 2 vCores and non-production consists of 4 vCores ESB Runtime worker available on CloudHub. It is recommended Fatface to buy 2 more production CloudHub licenses to be able to deploy all integration applications in scope.

From the infrastructure perspective, in future, when the number of systems increase and as per the business growth, the infrastructure can be upgraded when required. The chosen architecture using CloudHub and Anypoint platform which supports Vertical and Horizontal scalability, so the number of workers and their capacity can be always increased without any impact on the application.

Following link gives an overview on how applications can be scaled in CloudHub.

<https://docs.mulesoft.com/runtime-manager/cloudhub-fabric>

## Reliability

The runtime components are insulated from external network outages since they store a local cache and can continue to operate even if the Anypoint Platform were to become unavailable. Regardless, MuleSoft maintains an SLA of 99.99% for Anypoint Platform for APIs.

**Reliability during integration with Demandware and other Business Systems:**

It is assumed that all the systems ensure 99.99% availability and it is reliable. For any guaranteed message delivery requirements, Mule will use Amazon SQS. Please refer [implementation view](#_Implementation_View) for details on how Amazon SQS will be used in process and system APIs.

## Security

The MuleSoft Anypoint Platform can secure APIs or Mule Services using:

1. Transport security such as TLS
2. Authentication and token-based protocols such as SAML, oAuth, WS-Security.
3. Digital signatures for integrity and crypto algorithms for data encryption.

The platform allows security to be de-coupled from the implementation through the Anypoint Platform for APIs (API Manager), allowing policies to be applied and managed independently from code. These policies include:

1. Client ID Enforcement
2. oAuth2
3. SAML via Custom Policies
4. JWT via Custom Policies
5. IP Blacklisting and Whitelisting
6. Throttling & Rate Policies

It is recommended Fatface to expose all their experience API as HTTPS API with client-id security enforcement. All process and system APIs should be exposed privately as HTTPS API with client-id security enforcement. In order to achieve this all experience APIs will use port 8082 which by default uses the CloudHub LB and gets exposed over port 443. All process and system APIs should use port 8092 which can only be accessed in the Fatface private network and load balanced using DNS round robin.

**Security configurations:**

All the configuration details i.e. property files with external system credentials are secured and bundled in the Mule projects and this will be hosted on CloudHub which requires an authentication. These configurations are accessible only to the Anypoint platform admin and release/support team. Secure application properties will be used for deployment into higher environment such as UAT and Production. Refer below link for more details on secure application properties.

<https://docs.mulesoft.com/runtime-manager/secure-application-properties>.

## Auditing

Audit being a cross-cutting concern, it will be designed as a reusable and pluggable component using Mule flows and default logger component.

The considerations for auditing are:

* Works asynchronous on a low priority thread instead of running on high priority without disturbing the main thread of execution.
* In case if the Audit of request and response fails, it does not have any impact on the main thread of execution.
* Audit data will be logged into default log files of the application in CloudHub. Logs can be up to 100 MB or up to 30 days, whichever limit is hit first. This will be configurable so can be changed if needed.
* Audit will log all incoming and outgoing messages for an API call.
* Audit will log batch processing status for every batch processes.

## Performance

As majority of the project consists of file based processing the performance depends on the average bandwidth available for the files to be processed. Further as per communication with business the file size provided by FATFACE for any of the interfaces won’t exceed more than 5 MB size. This will in turn have less impact in terms of performance.

If message size and volume increases and performance issues are observed, horizontal or vertical scaling may be needed.

## Exception and Error Handling and Alerting

Global Reference Exception strategy is used for exception and error handling.

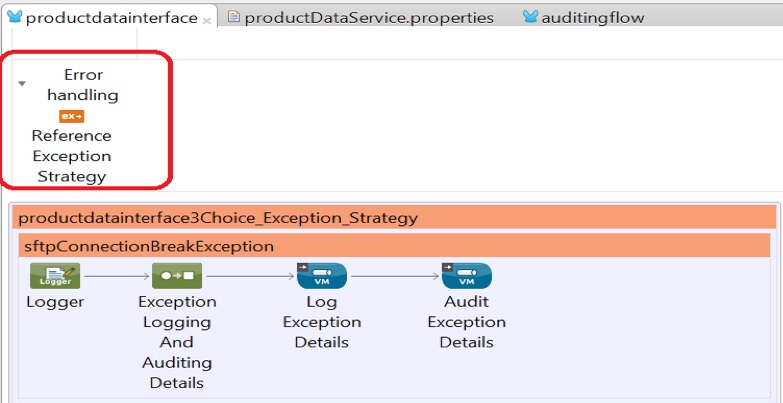
There will be a common Exception declared and all exceptions and error originated will be referenced or passed to the generic reference exception strategy.

Exceptions generated during runtime or transformation will be captured and logged via audit log under java.lang.Exception. This at any given moment of time will provide with the details of exceptions occurred.

Along with the Audit log entries in Database during the course of any exception the whole batch process will be retried for 2 more times (except for reading from SFTP location). So in the event of any exception the initial files will be stored in internal processing folder. The file will then again be read from the processing folder and processed accordingly. If still the exception or error happens after configured no. of retry attempts, then the file will be saved in SFTP location under the error folder section as depicted in section [4.3](#_Design_Solutions).

Inside Mule application Choice Exception Strategy is used to catch any exception/errors generated during the flow. Individual exceptions are internally caught using Catch Exception Strategy which is defined within the Choice Exception Strategy.

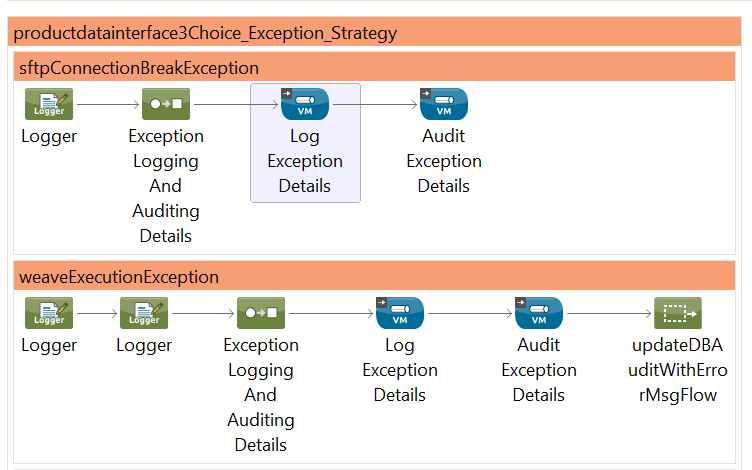
The Global Choice Exception Strategy for Product Data Interface is referenced inside a flow using Reference Exception Strategy.

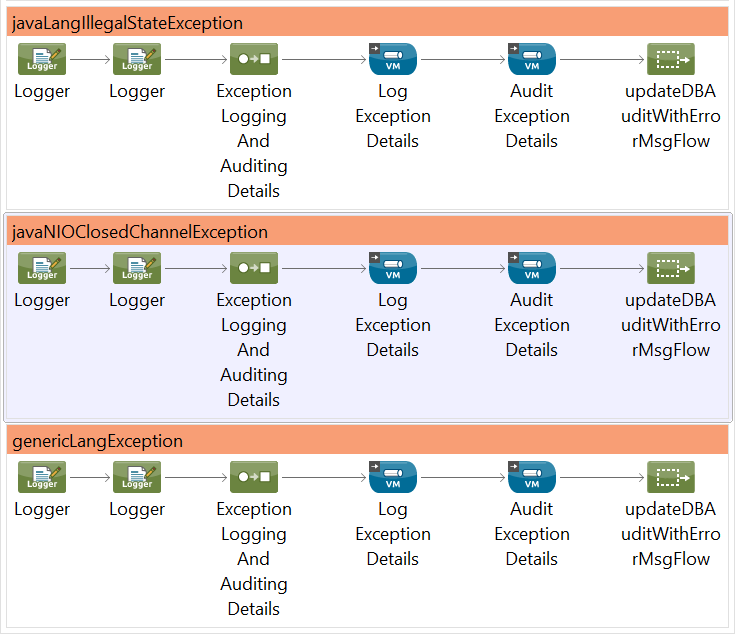


Some of the major exceptions handled are listed down below.

1. SFTP Connection Break Exception (java.net.ConnectException)
2. DataWeave Execution Exception (java.io.FileNotFoundException)
3. Java Lang illegal state Exception
4. Java NIO closed channel Exception
5. Generic Lang Exception where general exceptions are handled.

Snapshot for the same depicted below. Exceptions caught are handled accordingly and the exception details are logged in logger as well as audit trail in Database.





On Exceptions like SFTP connection break the process will simply log the exception details to logger and audit trail. In case of major exceptions other than logging and auditing the exception details, the ‘Start Error Flow’ will be triggered which will stop the process and clean any intermediary files generated and move to next file processing.



## Application Monitoring

The default out of the box dashboard and instance monitoring tooling available in MuleSoft Anypoint platform will be used for monitoring purpose. The Cloud console of the Runtime Manager displays [dashboards](https://docs.mulesoft.com/runtime-manager/monitoring-dashboards) with performance metrics for all applications deployed, both to CloudHub workers.

Additional API transactions details will be captured as part of auditing and logging will be used for future reporting.

Manual monitoring and health check will be part of the support activity carried out by the support team, so not in scope of this HLD.

## Batch Processing and Scheduling

A scheduled batch processing service that process the files for most of the interfaces and generates an XML output, depending upon the schema file and mapping provided. Following are the cron expressions that will be used for these interfaces.

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Summary** | **Frequency** |
| 1 | Updating store master data from Intraface to Demandware | Every Friday at 16:30 UK time |
| 2 | Initial load of Customer Data into Demandware | On demand |
| 3 | Order created in Demandware directed to ECommerceDW | Once every 5 minute  s |
| 4 | Initial load of Order History - from Venda to Demandware | On demand |
| 5 | Product data: CIMS to JDA | Once a day at 11:00 UK time |
| 6 | Pre-GRN to Future delivery: CIMS to JDA | Once every 5 minutes |
| 7 | Publish future date markdown price book from CIMS to Demandware | Once a day at 10:00 UK time |
| 8 | Update regular list price: CIMS to Demandware | Once a day at 10:00 UK time |
| 9 | Order update sent from CIMS to Demandware | Once every 5 minutes |
| 10 | Store inventory look up: Single SKU, Multiple Store | Realtime |
| 11 | NRT SKU inventory check from Demandware to CIMS | Realtime |
| 12 | Inventory updates from CIMS to Demandware | Once every 5 minutes |
| 13 | Order created in Demandware and sent to CIMS | Once every 5 minutes |
| 14 | Order Payment update details CIMS to Demandware directed to ECommerceDW | Once every 5 minutes |
| 15 | Order Refund /Cancellation request CIMS to Demandware directed to ECommerceDW | Once every 5 minutes |
| 16 | Order Status Changes: JDA to CIMS | Once every 5 minutes |
| 17 | Refund update notification: Demandware - CIMS | Once every 5 minutes |
| 18 | Payment update notification: Demandware - CIMS | Once every 5 minutes |
| 19 | Order cancellation: - CIMS to Demandware | Once every 5 minutes |
| 20 | Refund request notification: CIMS to Demandware | Once every 5 minutes |
| 21 | Take payment notification - CIMS to Demandware | Once every 5 minutes |
| 22 | Order Status Update CIMS to Demandware directed to ECommerceDW | Once every 5 minutes |
| 23 | Customer Order Returns: JDA to CIMS | Once every 5 minutes |
| 24 | Password Service | Realtime |
| 25 | GRN Receipt to CIMS | Once every 60 minutes |
| 26 | Product data: CIMS to InRiver | Once a day at 11:00 UK time |
| 27 | Inventory status changes: JDA to CIMS | Once every 5 minutes |
| 28 | New Order create: CIMS to JDA | Once every 5 minutes |
| 29 | Transaction Email Service | Once every 5 minutes |
| 30 | Daily full inventory update from CIMS to Demandware | On demand |
| 31 | Update marketing preferences in Demandware from XCM | Once every 60 minutes |

## Build and Deployment Solution

## Build Management

A Build tool is a tool that ensures the management of code dependency, compilation and linking, component managing, packaged, consistently, and reliably. Such tool can provide higher services such as managing the execution of unit tests and deployment.

Maven as a build tool will be used to standardize on build best practices, reduce dependencies, simplify continuous integration and easily share projects between developers.

## Continuous Integration

Continuous integration is the software engineering practice of frequently merging all developer working copies with a shared [mainline](http://en.wikipedia.org/wiki/Trunk_%28software%29). The key objective is to prevent code integration problems and improve standards by continuously building, testing and integrating code.

Jenkins is used as CI tool for FATFACE build.

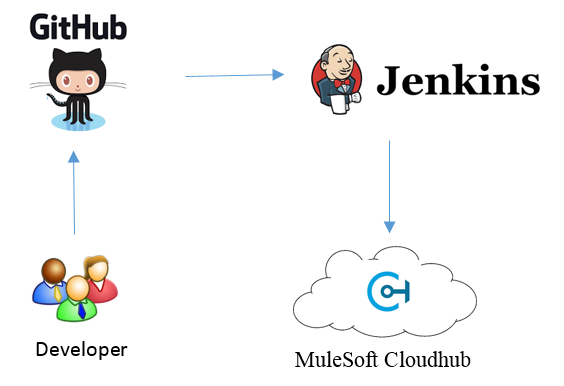


Figure . Continuous Integration

## Tools and Technologies

## Development Tools

1. Development IDE: Anypoint Studio 6.2
2. Development Platform: Windows

## Technologies

* Java 8
* Mule Expression Language
* XML/JSON
* Groovy
* SQL

## Environment

MuleSoft CloudHub runtime 3.8.3 with following environments created:

* DEV
* SIT
* UAT
* PRODUCTION

## Server Inventories

Below table provides a basic approximate configuration values for servers and applications that will be installed.

|  |  |
| --- | --- |
| Service | Amazon RDS |
| Purpose | Application database for Middleware |
| DB Engine and License | MySQL (Standard Support) or MS-SQL DB (Standard license - License Included or BYOL) |
| Environment | Production and Non-production |
| On Demand or Reserved? | On Demand |
| Throughput | 100 TPS with current data of 30k records Expected data growth is 100K records per year |
| Class | db.m4.large |
| Deployment | Multi-AZ (Primary/Secondary) |
| Number of Instances | 2 (Prod and Test) |
| Region | EU (Ireland) |
| DR | Not required |
| Data Size | 100 GB currently with 30% year-on-year increase |
| Storage Type | EBS |
| Backup Frequency | Daily – Differential and Weekly – Full |
| Backup Size (Storage Snapshot) | 300 GB per month approx. |
| Data Transfer Inbound | 1 GB per day (30 GB per month) |
| Data Transfer Outbound | 1 GB per day (30 GB per month) |
| VPC Requirements | Refer [Figure-11](#Fig11) to determine the VPC requirements |
| Additional Requirement or Remarks | This RDS instance will be used as a database for e-commerce project (and future integration projects) for staging any data for Mule applications and for auditing Mule API calls. |

|  |  |
| --- | --- |
| Service | Amazon EC2 |
| Purpose | Continuous Integration Server |
| Number of Instances | 1 |
| Instance Type | Linux/Ubuntu/Windows |
| Environment | Production and Non-production |
| On Demand or Reserved? | On Demand |
| Deployment | 1 |
| Class | m4.large |
| Region | EU (Ireland) |
| DR | Not required |
| HA | Not required |
| Data Size | 100 GB |
| Storage Type | EBS (General Purpose SSD) |
| Backup frequency | Daily – 2 Full and Weekly – 1 Full at any point of time, overall 3 backups will be available |
| Backup Size (storage snapshot) | 300 GB per month |
| Data Transfer Inbound | 25 GB per month |
| Data Transfer Outbound | 25 GB per month |
| No. of User Access to EC2 Box | 1 sudo user |
| VPC Requirements | Refer [Figure-11](#Fig11) to determine the VPC requirements |
| Elastic IP | Need one Elastic IP |
| Additional Requirement or Remarks | This EC2 instance will be used to install Jenkins, SonarQube & Maven repository (all these tools are open source and no commercial license will be required)  Jenkins will be used as a build server  Auto assignment of Elastic IP on reboot  SonarQube needs a backend DB. We will use Non-production DB for SonarQube |

|  |  |
| --- | --- |
| Service | EC2 |
| Purpose | Serv-U MFT SFTP Server |
| Number of Instances | 2 |
| Instance Type | Linux/Ubuntu/Windows |
| Environment | Production and Non-production |
| On Demand or Reserved? | On Demand |
| Deployment | Multi-AZ (HA required) |
| Class | m4.large |
| Region | EU (Ireland) |
| DR | Not required |
| HA | 2 nodes are required |
| Data Size | 100 GB |
| Storage Type | EFS |
| Backup frequency | Daily 1 parent and 1 baby backup |
| Backup Size (Storage Snapshot) | 100 GB per month |
| Data Transfer Inbound | 20 GB per day (600 GB per month) |
| Data Transfer Outbound | 20 GB per day (600 GB per month) |
| No. of User Access to SFTP | 10 users and 10 concurrent sessions |
| VPC Requirements | Refer [Figure-11](#Fig11) to determine the VPC requirements |
| Elastic IP | As per HA requirements |
| Additional Requirement or Remarks | This EC2 instance is for SFTP server Serv-U MFT Server is to be installed on this instance (license will be required) |

Note: Fatface is using Maytech SFTP. Same SFTP server is used by all applications such as MuleSoft, JDA, Demandware & Intrafce. Awaiting a document from Fatface to describe the actual SFTP server implementation so above table can be replaced.

# Testing







## Testing Approach

As a standard process of interface development, below testing would be conducted by WHISHWORKS team before the start of end to end testing (known as SIT2 phase):

1. Unit testing
2. Internal System Integration Testing (known as SIT1 phase)

Note: FATFACE Testing team would perform end to end testing on Mule Test Environment and WHISHWORKS would provide technical support for any issues reported during that period.

## Functional Testing

The functional testing is specific to the contracts that are established between various endpoints. Function Testing ensures that the transformation logic and validations are working as expected

Types of tests conducted:

1. Test for Data Validation
2. Test for Data Transformation
3. Test for Data Mappings

## Non-Functional testing

Non-functional testing would be carried out as per the agreed SLAs between various vendors like Demandware, CIMS, JDA, XCM and PIM. However, SLAs are yet to be finalized

Types of tests conducted:

1. Measure and Monitor transaction per second
2. Basic Authentication/Authorization (part of security) testing
3. Memory Leak Test
4. Load Test

## Testing Tools

The following testing tools will be used:

1. Munit for Unit Testing
2. SoapUI NG Pro (licensed by Fat Face) for parameterized testing of REST/SOAP call tests
3. Tool selection for evaluating performance can only be taken up once the SLA's and maximum concurrent users are identified.

## Mocking frameworks

RAML designing with any point API designer provides the mocking feature. The API portal provides mock services to check the test input and outputs. Mock services are available only during design phase

Munit is a Mule testing framework. It allows Mule developers to automate Mule applications testing in an easy manner. As a unit test framework, it provides the ability to mock different Mule components so that the test cases can be constrained and isolated.

# Licensing and Support

TBD

## Support Ownership

TBD

## Service Levels

TBD

## Licensing

* FATFACE is subscribed with Platinum subscription with 4 vCores Production and 4 vCores non-Production
* FATFACE owns GitHub cloud License
* FATFACE owns SOAPUI NG Pro License
* FATFACE is yet to buy Amazon SQS License

## Environment Management

TBD

\*\*\*End of Document\*\*\*