**Architecture Blueprint**

**Mulesoft Integration Architecture Design**

|  |  |
| --- | --- |
| **Prepared by:** |  |
| **Solutions Architect:** |  |
| **Programme Director:** |  |
| **Programme Business Owner:** |  |
| **Version:** |  |
| **Date:** |  |

TABLE OF CONTENTS

[1. Document Control](#_i0g5e5u6nmih)

[1.1. Document History](#_dxjk4yixaq4j)

[1.2. Document Sign Off](#_thdm46k6g5kq)

[1.3. Distribution List](#_vpj124fg9dpf)

[1.4. Related Documents](#_77i8vxwcg7c3)

[1.5. Glossary](#_uc04eo2xx8x5)

[2. Introduction](#_4skrkk7gim2j)

[2.1. Executive Summary](#_krki2dgeg1ih)

[2.2. Purpose of this document](#_x6zme56wpyzg)

[2.3. Intended Audience](#_gbqjzwkbgvqk)

[2.4. Project Overview](#_4whpskh2eudm)

[2.4.1. Historical Context](#_fup0bgriwi8)

[2.4.2. Business Purpose](#_d78e3517f1u0)

[2.4.3. Vision, Strategy and Commitment](#_tokc975jinzm)

[2.4.4. Target Customers](#_40xfv1v6feju)

[2.4.5. Operational Landscape](#_hasv33yuz0jh)

[2.5. Solution Boundary and Scope](#_zcv2e9xyynvg)

[2.5.1. In scope business processes:](#_6eesafw3tykv)

[2.5.2. Out of the scope:](#_t9zrytlwja8r)

[3. Architectural Requirements](#_loxe785clh9r)

[3.1. Functional Requirements](#_2w5wg2j3cbps)

[3.2. Non-Functional Requirements](#_iuz8i6thv0d0)

[3.2.1. Reusability](#_lu4tqmvz3v3z)

[3.2.2. Security](#_o8ilgvbytf4b)

[3.2.3. Performance](#_q7gsmarqu7fz)

[3.2.4. Volumes](#_6gou7p51o85a)

[3.2.5. Logging](#_s2271t4xqg7w)

[3.2.6. Auditing](#_mvlrypbuoi1p)

[3.2.7. Scalability](#_f1g0224aoqjs)

[3.2.8. Reliability](#_qmeqmu5prql8)

[3.2.9. Monitoring](#_x23wjgotpmm)

[3.2.10. Exception Management](#_lsryej7f78dw)

[3.2.11. Archiving](#_300v5an0o21y)

[4. Solution Architecture](#_nu008ufp31gy)

[4.1. High Level Design](#_e2ccdim5r98w)

[4.1.1. Inbound Channel (Installation, Device and Trader Data flows)](#_8r16vagao4w4)

[4.1.2. Outbound Request Channel (Report Generation Data flow)](#_14ttlzm3unht)

[4.1.3. Outbound Response Channel (Report Outbound Data flow)](#_kht6sc3y59xw)

[4.1.4. Consumption Data Flow](#_nwpp5p2zsa7x)

[4.2. Integration Business Process View](#_vlmp0zth0b9d)

[4.2.1. Inbound Integration Business Process](#_x9h45dbc803u)

[4.2.2. Outbound Integration Business Process](#_ghsbz1bygmdu)

[4.3. Application & API Led View](#_n9mr086n979p)

[4.3.1. Mule Application Landscape](#_qc9jfe8r4l2)

[4.3.2. Application Functionality](#_13catweyvy6k)

[4.3.3. Utility Applications and Frameworks](#_rzafijfap49v)

[4.3.3.1. Logging Framework](#_pbqpkilbat2v)

[4.3.3.2. Auditing Framework](#_mnfqy0ybkk9n)

[4.3.3.3. Business Alerts Framework](#_g7tow3gkn19p)

[4.3.3.4. Technical Alerts Framework](#_waf8kzv5gknj)

[4.3.3.5. APP1-File Util](#_lmdd9e8l03sm)

[4.3.3.5.1. APIs Exposed](#_a7k1xpbwysdr)

[4.3.3.5.2. Functionality / Flows](#_2t8yf78s48fc)

[4.3.3.6. APP2-Sequencer Util](#_1tbfux7q0sqx)

[4.3.3.6.1. APIs Exposed](#_edjwl51lenii)

[4.3.3.6.2. Functionality / Flows](#_usnjy3ks8dwb)

[4.3.4. System APIs](#_liir5dy80e9)

[4.3.4.1. APP3-Sys-sap](#_mu02pk34giuh)

[4.3.4.1.1. APIs Exposed](#_bhmzi61cqnd6)

[4.3.4.1.2. Functionality / Flows](#_zdhy93lj0189)

[4.3.5. Process APIs & Orchestration Applications](#_pe6u5wof3eu5)

[4.3.5.1. APP6-Proc-assets-mng](#_nesy6sff854j)

[4.3.5.1.1. APIs Exposed](#_q7lszq13r9br)

[4.3.5.1.2. Functionality / Flows](#_5ju90pypufzb)

[4.3.6. Experience APIs](#_99hijbyu1xx0)

[4.4. Application Interaction and Functionality View](#_bsbqa05ybyx6)

[4.4.1. Inbound Interactions - SAP/COUNTIES](#_37i3i0y8qm53)

[4.4.4. Outbound Interactions - Consumption Report](#_36hra1n3wu5l)

[4.5. Data and Information View](#_85wn2xvgw9m3)

[4.5.1. SAP Data model](#_ausebpr6kdcy)

[4.6. Deployment and Physical View](#_o7c18kfib9u7)

[4.6.1. Environment: TST1](#_9adeb3lfcfl4)

[4.6.2. Environment: TST2](#_llt8ri3g453q)

[4.6.3. Environment: PRE-PROD (UAT)](#_nhltkrh9hg2v)

[4.6.4. Environment: PROD](#_5ugy0pylq319)

[4.6.5. Environment: DR](#_gazrua3mhsr0)

[4.7. Software Versions](#_626xk5huunw8)

[4.8. Transports and Protocols View](#_1n4brygs7x8q)

[4.9. Security View](#_bhaw6v34gwve)

[4.9.1. Inbound Security Enforcement](#_sa3ibxli4dvp)

[4.9.1.1. Authentication](#_f87op95o9uah)

[4.9.1.2. Authorization](#_k8wju26sn12z)

[4.9.1.3. Encryption (SSL)](#_c6rigz6ox0u6)

[4.9.2. Outbound Security Configuration](#_e6th1u1rtij7)

[4.10. Persistence Store View](#_aqqp8af78pj1)

[4.11. Error Handling View](#_ovdv2v37dlc3)

[5. Implementation Guidelines](#_cs6quvie0ihs)

[5.1. Logging and Auditing](#_75jrekbtfrqz)

[5.1.1. Definitions](#_ue4mxarvi90x)

[5.1.2. Key Considerations and Best Practices applicable to Logging and Auditing](#_herjfx2wte4x)

[5.2. Naming Conventions](#_xrjv912eo0e)

[5.2.1. Artefacts and Components naming conventions](#_1rmbf3b39l33)

[5.2.2. Deployable Mule Application Archive Naming Convention](#_1rvwp1q)

[5.2.3. Source Code Best Practices](#_116lyifg5k1n)

[5.2.4. Continuous Integration](#_jflss21tizoj)

[6. Mule Apps Configuration and Migration](#_9y3eiq9eyawn)

[7. Architecture Registers](#_v3pax9cfb82c)

[7.1. Decisions](#_m4mk6kfqlx4)

[7.1.1. About Error Handling Notifications](#_f32nri347ffy)

[7.1.2. About the usage of Database instead of JMS](#_1kbqryo3p4fq)

[7.1.3. About the decision of using realtime web services vs asynchronous for the Inbound pattern.](#_g9eq6zh6aszt)

[7.1.4. About the outbound pattern to collect file pieces from MDM and send the file](#_s2vpkofx8u1t)

[7.1.5. About the usage of 1 single JMS queue out of MDM](#_v3dhammhsdl2)

[7.1.6. About the technology of the JMS Broker](#_94qnpijo7r7w)

[7.1.7. About the attributes of logging](#_iox8kqvbpfr5)

[7.1.8. About the attributes of auditing](#_nrfmewsniro5)

[7.1.9. About the destination of logging](#_eqvkecxp94lp)

[7.1.10. About the destination of auditing](#_j5ef2f3j88us)

[7.1.11. About the Primary Keys](#_5877phvk1syr)

[7.1.12. About the new format of SAP](#_qiqilv2w1zj0)

[7.1.13. About the lookups in Counties](#_9iwp55kcfzyc)

[7.1.14. About the real time call to the registry using the API instead of the Web service](#_67kkbo57f8f4)

[7.1.15. About the Mule Management tool](#_inhp50ccif9k)

[7.2. Risks](#_hu29hm6y2toc)

[7.3. Assumptions](#_3l18frh)

[7.4. Issues](#_7jlsg5vb5969)

[7.5. Dependencies](#_58sblpv4075q)

[7.6. Constraints](#_madxd6uejk9d)

[8. Appendix - API Led Approach](#_6qeh7qj4zqb4)

[8.1. Introduction](#_boikw7vy8a87)

[8.1.1. “Three-layered” API-led connectivity architecture](#_qpgezjinkgd1)

[8.1.2. Benefits of API-led connectivity](#_v5olc4v1jbl4)

[8.1.2.1. Business Benefits](#_7vqpp3wq7h51)

[8.1.2.2. Technical Benefits](#_lxumcqlceevq)

[9. Appendix. Environment Configuration Details](#_8pyvgs9n404s)

# 1. Document Control

## 1.1. Document History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Version Date** | **Updated by** | **Description of Changes** |
|  |  |  |  |

## 1.2. Document Sign Off

### Version 1.0

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Role** | **Sign-Off Date** | **Signature** |
|  |  |  |  |

### Version 2.0

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Role** | **Sign-Off Date** | **Signature** |
|  |  |  |  |

## 1.3. Distribution List

|  |  |
| --- | --- |
| **Name** | **Role** |
|  |  |

## 1.4. Related Documents

|  |  |
| --- | --- |
| **Related Document** | **Link** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## 1.5. Glossary

|  |  |
| --- | --- |
| **Term** | **Description** |
| ESB | Enterprise Service Bus |
| MMC | Mule Management Console |
| JVM | Java Virtual Machine |
| OOTB | Out Of The Box |
| DLQ | Dead Letter Queue |
| RAML | RESTful API Modelling Language |
| HA | High Availability |
| DR | Disaster Recovery |

# 2. Introduction

## 2.1. Executive Summary

The MuleSoft Anypoint Platform will be used by *<COMPANY NAME>* as an API-led Integration Platform layer to connect to multiple on-premise and cloud based systems, as part of the *YYY* project. There are over *n* different potential integration use cases that need to be managed across all systems, with each Mule application having multiple data type exchanges.

Mule Applications are created in the MuleSoft Anypoint platform to incorporate integration & business logic processes. This document is to provide details regarding the following:

* Reviewing and documenting functional and non-functional requirements
* Developing high-level architecture and design to meet the requirements.
* Provide recommendations for the implementation plan
* Provide best practices for implementing Mule solution

This will significantly reduce the risks by ensuring the solution is designed right the first time and minimise the need for additional rework in the development phases and reducing the delivery time through quicker solution delivery leveraging the defined Best Practices

## 2.2. Purpose of this document

This document provides a comprehensive architectural overview of the systems used within *<COMPANY NAME>*. This document will provide the final high-level architecture and the detail around the various components of the architected solution.

## 2.3. Intended Audience

The target audience for this document includes *<COMPANY NAME>* Project Managers, Enterprise Architects, Solution Architects, and Development teams.

## 2.4. Project Overview

### 2.4.1. Historical Context

*<COMPANY NAME>* core business is *<describe core services>*

*<COMPANY NAME>* is part of *<describe company structure>*

### 2.4.2. Business Purpose

The current program “*YYY*” main goal is to provide *<describe company’s business goals>*

### 2.4.3. Vision, Strategy and Commitment

The current program tries also to establish the first initiative in *<COMPANY NAME>* to provide*<vision and strategy of project>*

It has been identified that the role of MuleSoft technologies as part of this initiative is very important, because it can act as a key enabler of Data Services and Client Experience, providing Data services as API’s aligned with API-Led Connectivity approach.

### 2.4.4. Target Customers

Currently, *<COMPANY NAME>* customers are around *<describe business’ customer base and target customers>*

### 2.4.5. Operational Landscape

*<Description of business landscape, such as, service numbers, locations, operational functions>*

## 2.5. Solution Boundary and Scope

### 2.5.1. In scope business processes:

The integration solution presented in this document covers the integration that supports these main business processes:

* + *<List of integration requirements between in-scope systems>*

### 2.5.2. Out of the scope:

* *<List of out-of-scope requirements>*
* Any other element outside of the elements considered in scope will be considered “out of the scope”.

# 3. Architectural Requirements

The following sections describe the functional and nonfunctional requirements, at a high level, that are part of the solution.

## 3.1. Functional Requirements

The following table summarises the functional requirements that are going to be implemented in the solution. The last column, “Integration Involved”, clearly specifies if that functional use case scenario is going to be delivered as part of the Mule integration or not (only the ones with value “Yes” are considered in the Mule scope).

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Scenario** | **Process Summary** | **Data Sets (payloads) required** | **Integration Involved** |
| (1) |  |  |  |
| (2) |  |  |  |
| (3) |  |  |  |

## 3.2. Non-Functional Requirements

The following NFR’s describe the requirements that the Mule platform will align.

### 3.2.1. Reusability

As a principle and a requirement, all the APIs delivered by the Mulesoft platform should follow the reusability principle in order to align with the API-Led strategy.

### 3.2.2. Security

The security requirements collected are:

|  |  |  |
| --- | --- | --- |
| NFR01 | Roles and Permissions |  |

|  |  |  |
| --- | --- | --- |
| NFR02 | User Management |  |
| NFR03 | Security and encryption |  |

### 3.2.3. Performance

|  |  |
| --- | --- |
| System Performance | *<Example: Confirm that the system performance is acceptable as per the criteria below*   * *CPU usage to not exceed 80% for more than a 2 hour period* * *CPU usage to be managed so that access to and the management of Mulesoft is never compromised and does not impact stability of the Mulesoft platform.* * *Database response < 12ms*   *Other system criteria to be monitored*   * *Memory usage* * *Disk and Database I/O* * *Disk space* * *SAN and network traffic*   *>* |

### 3.2.4. Volumes

These are expected averages of volumes to be managed by the platform. They don’t consider large scale deployment, upgrades or replacements. For those situations, another volume table will be applicable and new sizing calculations will be required.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Use Case | Name | Volume | Type | Per |
|  |  |  |  |  |
|  |  |  |  |  |

### 3.2.5. Logging

*<Example: Logging of all exceptions inbound and outbound to Mulesoft including*

* *Transformations*
* *Errors*
* *Notifications*

*Logs should be retained for a determined period in order to add supporting the solution component. After which rotation, purging or archiving of the logs should be considered based on the requirements of the solution component.*

*>*

### 3.2.6. Auditing

To ensure *<COMPANY NAME>* meets regulatory compliance the following audit capability is required:

*<Example: In all situations where raw consumption and event data is processed (e.g. reprocessing where an exception has occurred, event filtering, data validation, estimation or edited by a user) an audit history must be kept. The audit history must contain the following:*

* *The original raw consumption and event data*
* *The record or detail (who, when, why) of the change which occurred*
* *The new consumption and event data*

*>*

### 3.2.7. Scalability

The solution should allow horizontal scalability of the HA cluster to allow at a given point in time the addition of new processing nodes to increase processing capabilities.

The Scalability of the Mule data store is expected to be handled by the *ZZZ server* and services outside of Mule's control.

### 3.2.8. Reliability

|  |  |  |
| --- | --- | --- |
|  | Down Time | *<Example: YYY* infrastructure must achieve a 99% availability rate, systems are considered to be highly available with provisioned maintenance windows to minimise downtime. MS will have an HA cluster which makes the solution Highly Available to guarantee minimised downtime.> |
|  | Infrastructure Backups | *<Example: The system file structures / databases will require system backups and are relying on current <COMPANY NAME> systems to provide this.>* |
|  | System Backups |  |

|  |  |  |
| --- | --- | --- |
|  | Disaster Recovery | *<Example: All systems should be included in the Disaster Recovery (DR) solution for <COMPANY NAME>*  *MS will have a DR installation . An inactive cluster will be set up in the DR site matching the configuration of Production .>* |

### 3.2.9. Monitoring

|  |  |  |
| --- | --- | --- |
|  | Infrastructure Monitoring | *<Describe monitoring requirements>*  Anypoint Runtime manager Out of the box capabilities will be used to monitor applications . |
|  | System monitoring | *<Describe monitoring requirements>*  Monitoring of the backend database / Memory usage for Mulesoft as above |

### 3.2.10. Exception Management

In case of exceptions, the requirement to handle them is as follows:

* **Business Exceptions** (related with data quality or business rules violated): An entry should be logged into … in order a … user take corrective actions.
* **Technical Exceptions** (related with technical conditions and error -connectivity, security, etc..-): An entry should be injected into … in order for System Analyst to take corrective actions.

### 3.2.11. Archiving

To ensure <COMPANY NAME> meets regulatory compliance in the delivery of … the following archiving capability is required:

*<Examples:>*

* Raw consumption and event data processed in the last 92 days (to support the market reconciliation period) is available to <COMPANY NAME> Operations immediately.
* All raw consumption and event data and exception data transfer containers (files, messages, etc.) used for delivery from the meters through <COMPANY NAME> systems to the customer are stored for 2.5 years and can be made available to <COMPANY NAME> Operations within 1 business day.
* All raw consumption and event data and exception data transfer containers (files, messages, etc.) processed in the last 7 years (outside of 2.5 years) is available to <COMPANY NAME> Operations within 5 business days.
* Where transfer containers (files, messages, etc.) cannot be edited or changed and this is a robust and demonstrable process the transfer containers (files, messages, etc.) do not need to be stored.

# 4. Solution Architecture

This section describes the solution at the architecture level, covering different points of view that address different concerns regarding architecture aspects.

## 4.1. High Level Design

*<INSERT HIGH LEVEL DIAGRAM>*

The High Level Design (HLD) describes a very high level of the architecture of the solution, as well as the main systems and information flows that implement the use cases of the solution:

The information flows that are part of the scope of these solutions are numbered as follows:

*<INSERT HIGH LEVEL DESCRIPTIONS OF INFORMATION FLOWS>*

## 4.2. Integration Business Process View

The following diagrams illustrate the integration business process happening in Mule for the Inbound and Outbound data flows.

### 4.2.1. Inbound Integration Business Process

*<INSERT DIAGRAM>*

*<INSERT DESCRIPTIONS>*

### 4.2.2. Outbound Integration Business Process

*<INSERT DIAGRAM>*

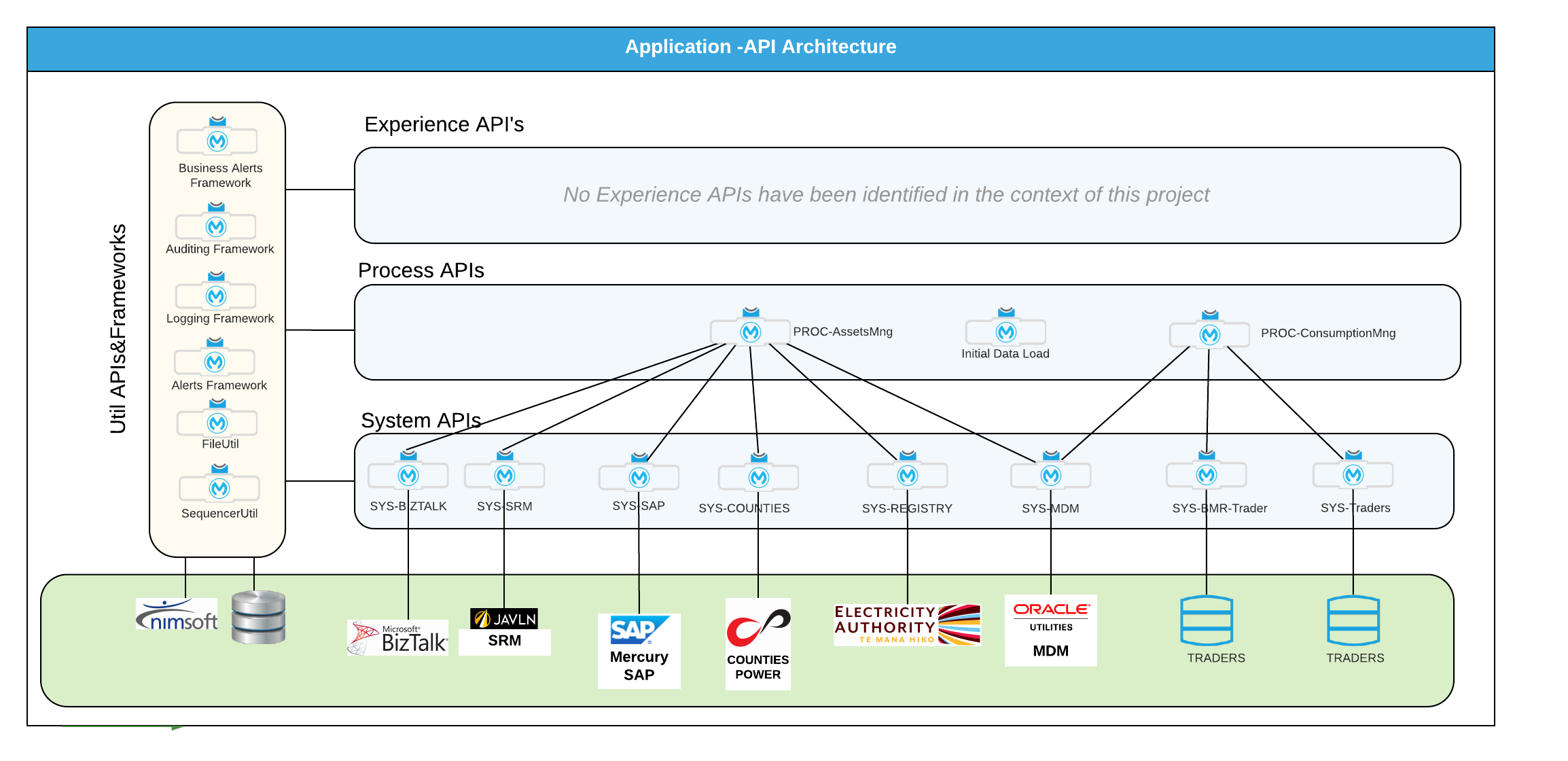
*<INSERT DESCRIPTIONS>*

## 4.3. Application & API Led View

The purpose of this section is to provide enough level of detail of the mule applications and the functionality of those applications in order to design and build a coherent API - led approach for the ICE 2 Project, as well as having them as a reference for future reuse and reference.

### 4.3.1. Mule Application Landscape

*<INSERT DIAGRAM, Example:>*



In the previous picture we can identify the mule applications that are going to be delivered as part of this project.

The applications are placed in 4 different categories, depending of their function aligned with the API Led approach:

* **Utilities and Frameworks**: Under this category, there are common services, components and API’s that are going to be used across the whole solution, providing a unified way of accessing and using that common functionality. The components and APIs identified are listed in the following table:

|  |  |  |
| --- | --- | --- |
| Component/Application Name | Type | Function |
| Business Alerts Framework | Component | Sends alerts related with business situations to the business users. |
| Auditing Framework | Component | Sends auditing information related with the execution of the flow in Mule |
| Logging Framework | Component | Sends logging information related with the execution of the flow |
| Alerts Framework | Component | Sends technical alerts to the system monitoring platform: |
| util-file | Application | Manages the storage and the retrieval of files across the whole solution. |
| util-sequencer | Application | Resolves the sequencing problem for files and messages that have to be processed in certain order |

***Note****: A Component is a drag-and-drop building block that is contained in applications. A component in Mule also can be called a message processor.*

* **System APIs**: Are those applications that provide an abstraction layer between the systems of records and the integration layer. The System APIs unlock the system data by exposing it through standard API HTTP REST interfaces. Therefore, this abstract layer eliminates the complexity of accessing those system data. The applications playing the role of System APIs are listed in the following table:

|  |  |
| --- | --- |
| System | Mule Application Name |
| <System name, e.g.>  SAP | <Mule Application name, e.g.>  sys-sap |
|  |  |

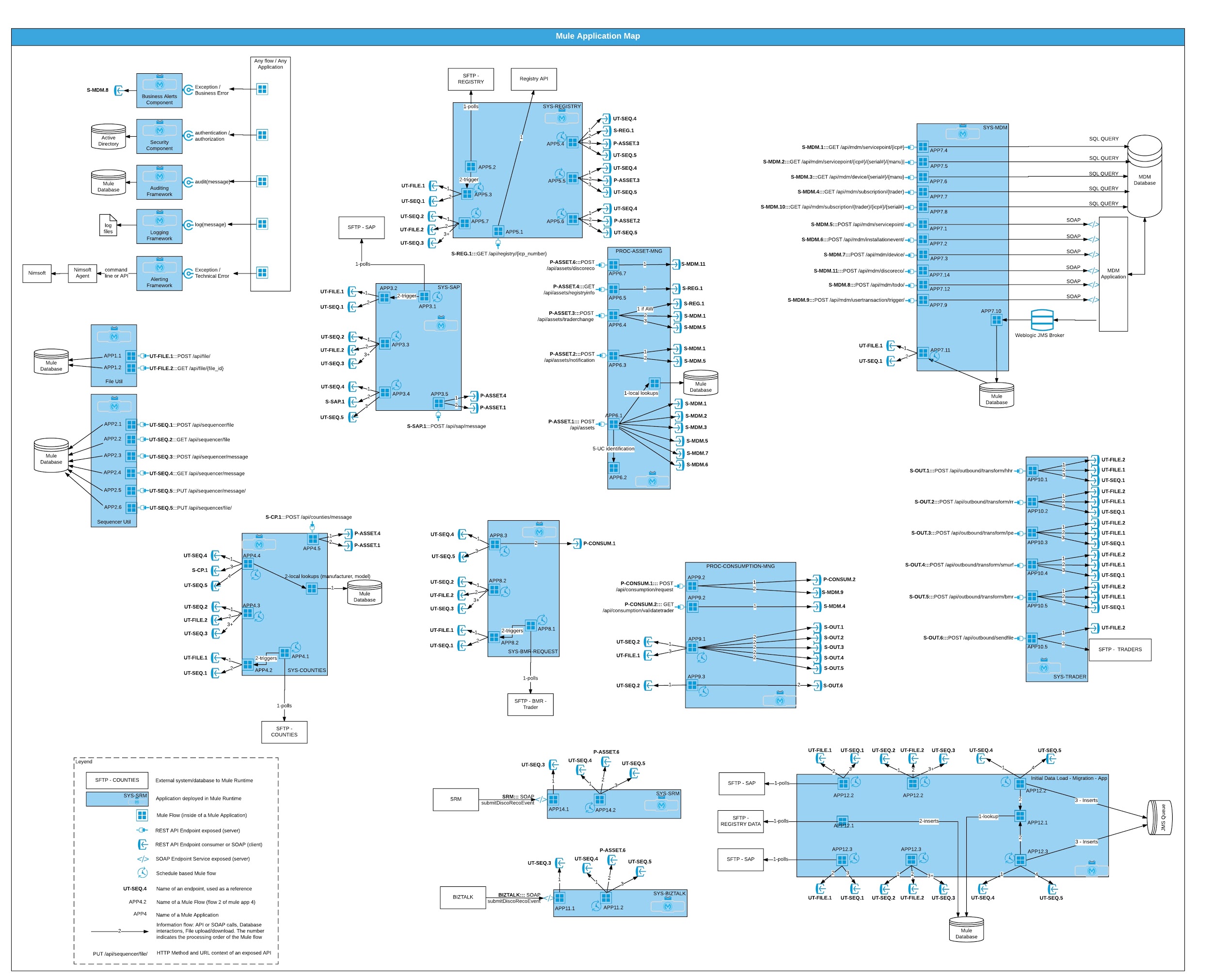
* **Process APIs**: Those applications that provide the orchestration and the composition of the system API’s. In these applications, the business logic is implemented, as well as system-agnostic rules that are more business-related. There are 2 Process APIs identified:

|  |  |
| --- | --- |
| Process | Mule Application Name |
| <Process name, e.g.>  Inbound Asset Management Process | proc-asset-mng |

### 4.3.2. Application Functionality

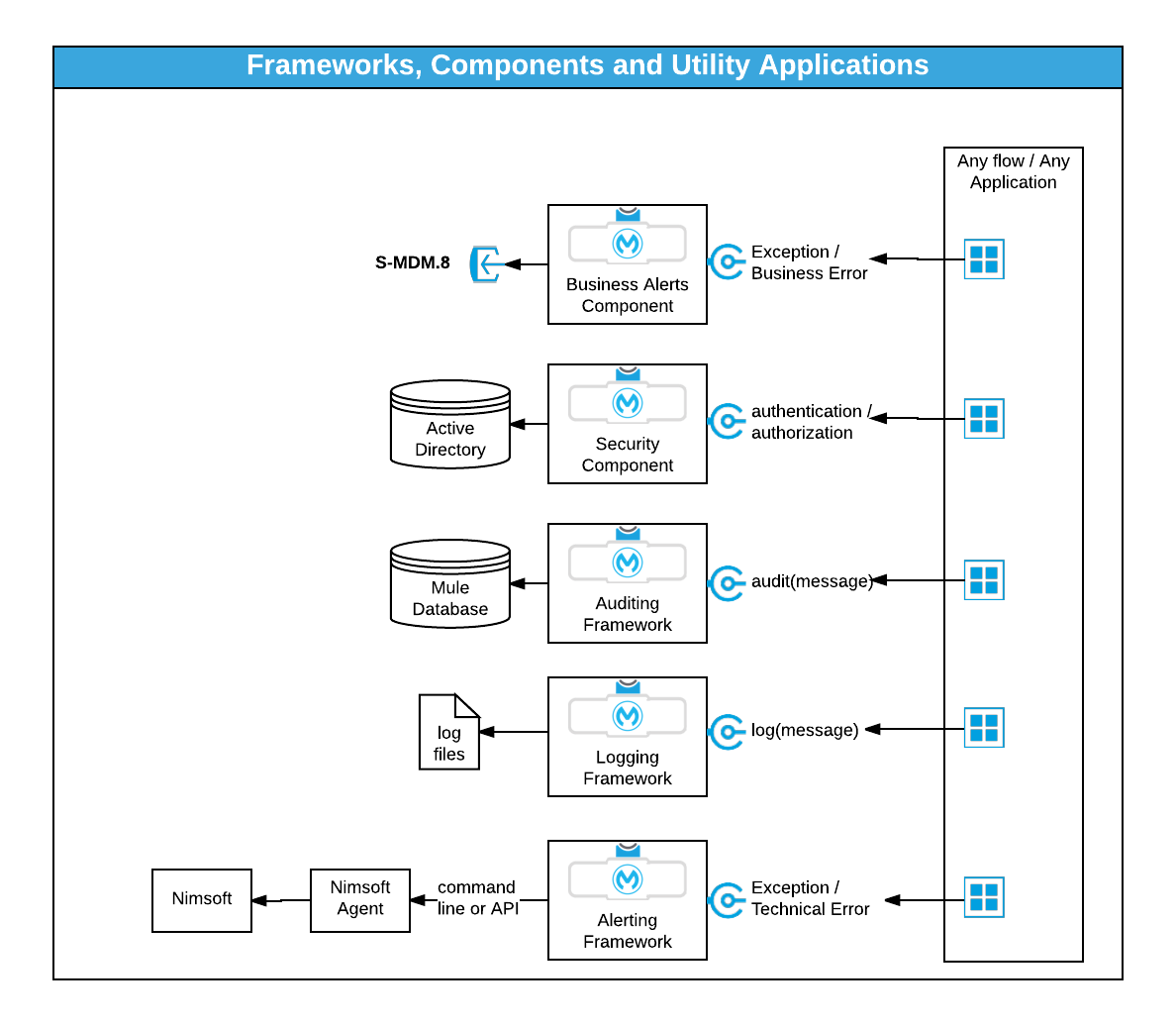
This section describes the APIs and the functionality that every application exposes, in order to understand the context of its role in the whole solution.

The following diagram depicts the whole application functionality, from a static point of view, of the solution:



### 4.3.3. Utility Applications and Frameworks

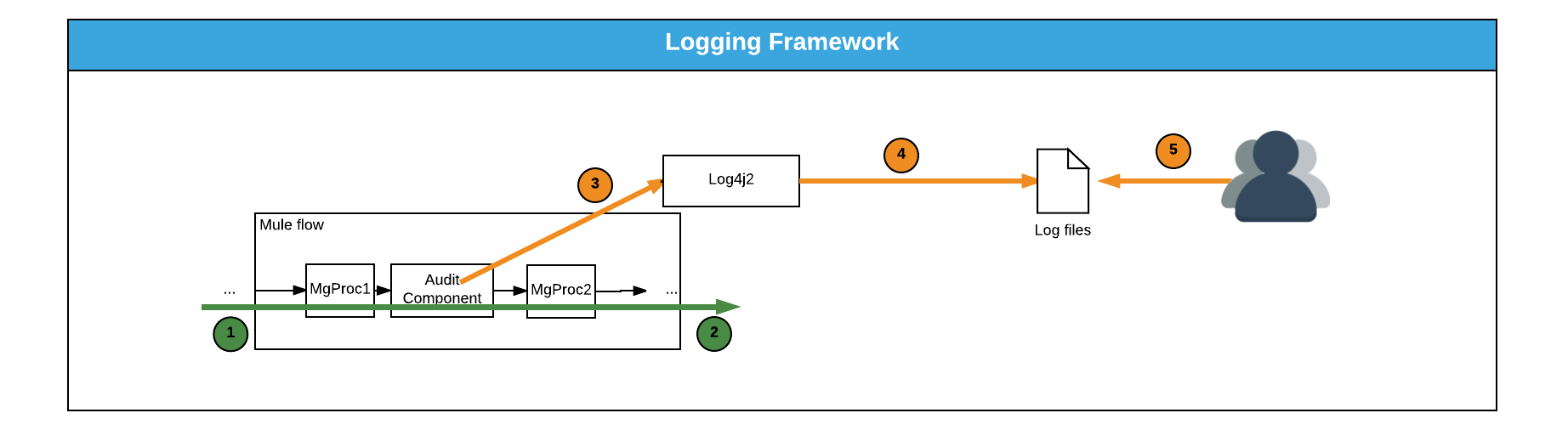
An overall diagram of these frameworks can be seen here:



#### 4.3.3.1. Logging Framework

The logging framework will be composed by a component that will be placed in the Mule flows. This component (message processor) will log messages into log files with a particular structure, in order to make the log messages consistent across the whole solution.

This framework is based on the apache log4j2 libraries:



1. The message enters in a mule flow and it is executed by the message processors chain
2. The message exits the mule flow.
3. At certain point, a log component is placed in the flow and the log message is routed to the logging framework.
4. The logging framework, based on the Log4j2 component will receive the message and, asynchronously, the message is routed to the file appender. The log message is written in the log file.
5. An operator can now open the file and see the log messages.

The information that this framework is going to be able to log is present in the following table:

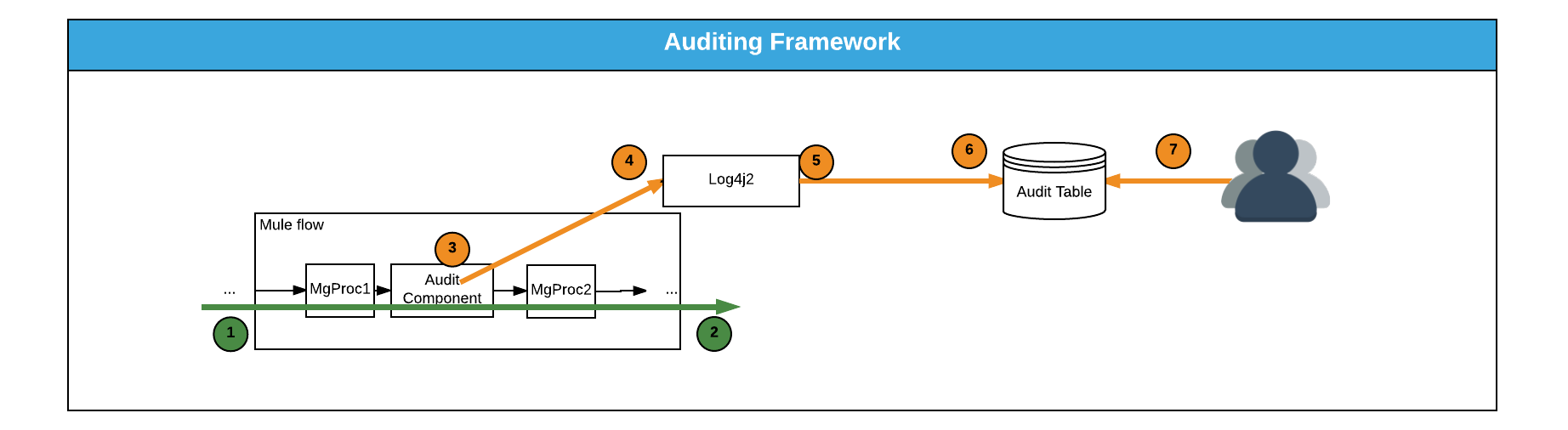
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Category | Field Name | Field Type | Description | Example |
| Header | Audit id | String | End to end transaction id | 12934823n1h8d2833hh3-23493 |
|  | Type | String | The event which is being logged  Possible Values:  “Inbound” (for calls received from a Client platform) OR  “Outbound” (for calls to downstream platforms or data sources) OR  “Batch” (for calls from internal components like daemons or schedulers) OR  “Security” (for security and compliance events, for details see Appendix for Application Security Logging) OR  “Security\_InfoAccess” (for security and compliance events, for details see Appendix for Application Security Logging) OR  “Other” (for all other events) | Inbound, Outbound,  Batch,  Other |
|  | StartTS |  | The start time of the event.  For Inbound calls, this is the time the request was received.  For Outbound calls, this the time the request was sent to the downstream platform. | yyyy-MM-dd HH:mm:ss.SSS-zz:zz  y = year  M =Month  d = day of month  H = hour of day (0-24)  m = minutes  s = seconds  S = milliseconds  z = utc offset |
| Source | IP | IP address | IP address of the originating device | \* |
|  | MAC/Node | MAC address or Node | Mac address of the source | \* |
|  | SRC\_Runtime | String | Information about the calling context(s), in the form of a “user agent” string  Examples: IPhone;  WAS/6.1; Java 1.6.0/14b  IIS/8.0; CLR/4.0 | \* |
|  | SRC\_AppName | String | Application name on the originating platform  (example:SAP ,CP) | AppName |
|  | SRC\_AppVersion | String | The version of the application (Example 1.0) | Version # |
|  | SRC\_ReqId | GUID | The Request or Correlation Id received from the source (if available) | \* |
|  | APP\_Runtime | String | Information about the calling context(s), in the form of a “user agent” string  Examples:  Java 1.8.0/14b | \* |
|  | APP\_AppName | String | Name of the application that is issuing the log. | AppName |
|  | APP\_AppVersion | String | The version of the application based on deployment versioning | Version # |
|  | Host | String | The host on which the event is occurring | \*.\* |
|  | Channel | String | Mobile/Web/batch/SAP | \* |
|  | APP\_ReqId/transaction id | GUID | Correlation Id for the current unit of work on the application platform.  Example: "6dacbde2-525c-4a38-bfd0-0e924fe4db6c" | \* |
|  | URL | String | Full URL of the service call (including querystring parameters)  Example: https:// test.<COMPANY NAME>.co.nz/ | \* |
|  | CallType | String | Identifies the type of call (invocation style). Only elements of the list “Values” section can be used. | WebHttp, JDBC, ODBC, RPC, LPC, RMI, MQ, Rest, SOAP |
|  | Operation | String | Service Operation  Examples:  For Calltype REST: GET or POST  For CallType SOAP: "GetDeviceInformationl"  For CallType JDBC: INSERT or SELECT | \* |
|  | Port | String | Service Port Name | \* |
|  | BytesSent | Integer | Bytes sent in the service call. | \* |
| Results  (The results of the service call) | Elapsed | Integer (milliseconds) | Time from receipt of the incoming request, through to eventual response | Time in milliseconds |
|  | ErrorCode | Integer | All exceptions result in an error code. This field is present when TraceLevel = “Error” | \* |
|  | ExceptionType | String | The class of the exception. This field is present when TraceLevel = “Error”  Example: “System.TimeoutException” | \* |
|  | Message | String | Error, info or warning message issued by the service, prefixed by a Message Number | \* |
|  | TraceLevel | String | Information, Warning or Error | \* |
|  | BytesRet | Integer | Bytes returned by the service call. | \* |
| Actor | User | String | Required if manual processing | \* |
|  | System | String | ID of the system executing the transaction (if applicable) | \* |
| Security | Token id | String | The token id for the request | \* |
|  | SomeField1 | String | And extended security field. Field is required depending on SecurityLoggingSupplement | \* |
| Extended\_Fields | SomeField1 | String | Any field the application owner wants to add. Can be any Format |  |
|  | SomeField2 | String |  |  |

For more information about the logging framework, please refer to section “Implementation Guidelines - Logging and Auditing”

For more information about the capabilities of log4j2, please visit <http://logging.apache.org/log4j/2.x/>

#### 4.3.3.2. Auditing Framework

The auditing framework is very similar to the previously described logging framework. It is also a component (message processor) that will write messages into a audit database table with a particular structure, in order to make the audit messages consistent across the whole solution. Its usage is intended to capture business events, or valuable information that is going to be used within a business context, like security auditing, use case statistics, performance measurements or even other type of analytics:



1. The message enters in a mule flow and it is executed by the msg processors chain
2. The message exits the mule flow
3. At certain point, an Audit component is placed in the flow and the audit message is routed to the auditing framework.
4. The auditing framework, based on the Log4j2 component will receive the message.
5. Asynchronously, the message is routed to the database appender.
6. Write the content of the audit message to the AUDIT table in the database.
7. An operator can execute a query to aggregate and extract auditing information.

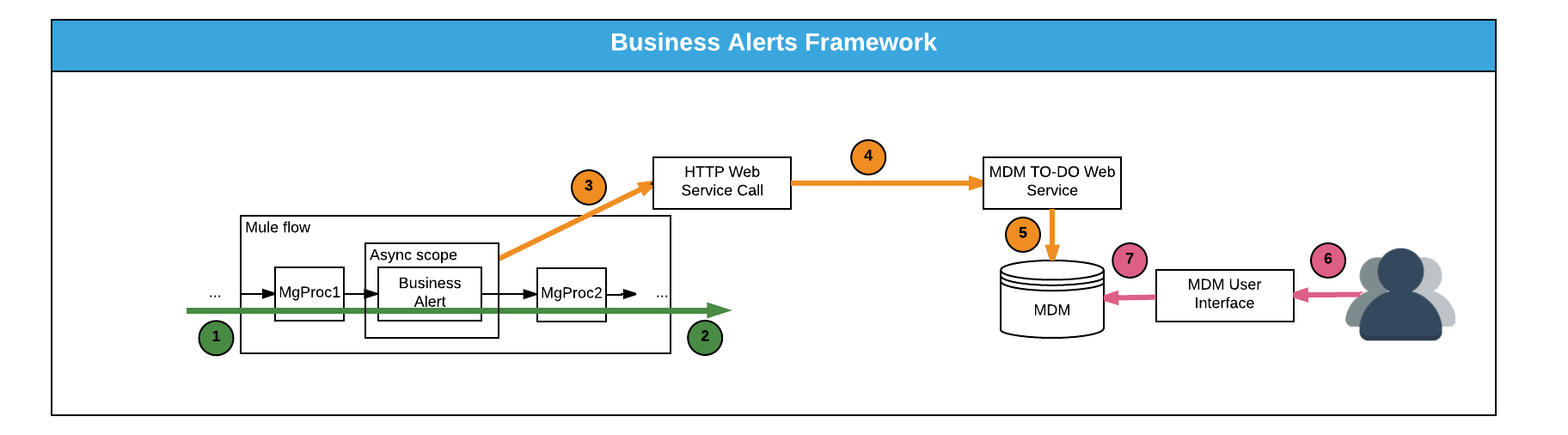
This framework is based on the apache log4j2 libraries, and the audit messages will be written into a database table.

The fields that are going to be part of an auditing message are:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| audit\_id | Numeric | Unique identifier of the audit event |
| audit\_key | String | level 1 of trace hierarchy. Used to trace flow across different events. Can be null |
| audit\_key2 | String | level 2 of trace hierarchy. Used to trace flow across different sub events with common audit parent key. Can be null. |
| audit\_key3 | String | level 3 of trace hierarchy. Used to trace flow across different sub events with common audit parent key 2. Can be null. |
| audit\_key4 | String | level 4 of trace hierarchy. Used to trace flow across different sub events with common audit parent key 3. Can be null. |
| audit\_timestamp | Timestamp  In the format of:  yyyy-MM-dd HH:mm:ss.SSS-zz:zz  y = year  M =Month  d = day of month  H = hour of day (0-24)  m = minutes  s = seconds  S = milliseconds  z = utc offset | timestamp of the occurrence of the event. |
| audit\_category | String | Category of this auditing event: Security, Performance, Business KPI, Business Trace, Business Process, Use Case, etc... |
| audit\_type | String | A type that this audit event belongs to: access, number of transactions, response timestamp, etc.. |
| audit\_name | String | A particular name of the event, descriptive, aligned with the category and type |
| audit\_pre\_data | String | Any data relevant with a pre evaluation, rule or step. |
| audit\_post\_data | String | Any data relevant with the result of the evaluation, rule or step. |
| audit\_data01 | String | Field to store generic data related with the category, type and name |
| audit\_data02 | String | Field to store generic data related with the category, type and name |
| audit\_data03 | String | Field to store generic data related with the category, type and name |
| audit\_data04 | String | Field to store generic data related with the category, type and name |
| audit\_data05 | String | Field to store generic data related with the category, type and name |
| audit\_data06 | String | Field to store generic data related with the category, type and name |
| audit\_data07 | String | Field to store generic data related with the category, type and name |
| audit\_data08 | String | Field to store generic data related with the category, type and name |
| audit\_data09 | String | Field to store generic data related with the category, type and name |
| audit\_data10 | String | Field to store generic data related with the category, type and name |

#### 4.3.3.3. Business Alerts Framework

In order to deliver messages to business users notifying about the occurrence of exceptions and cases where validations and business rules are returning an exception, the following component will be available:



1. The message enters in a mule flow
2. The message exits from the mule flow
3. During the processing of the message, there is an async scope that guarantees that the delivery of the business alert is going to be asynchronous, so 2) won’t be waiting for 3) to complete.
4. The component will execute a web service call to the MDM TO-DO Web Service.
5. The Web Service in MDM writes the content of the TO-DO entry in the database.
6. A user checks the TODO list in her/his user interface.
7. The MDM application will access to the TO-DO entry created by Mule platform.

***NOTE: It is very important to enclose the Business Alert component within an “Asynchronous Scope” in the mule flow to make sure that the call is executed asynchronously***

This component (message processor) will route all these business exception messages to MDM, using the functionality of MDM of TO-DO-List.

The way of injecting the information into the TO-DO-List is via a Web Service Call that MDM exposes as part of its out-of-the-box functionality.

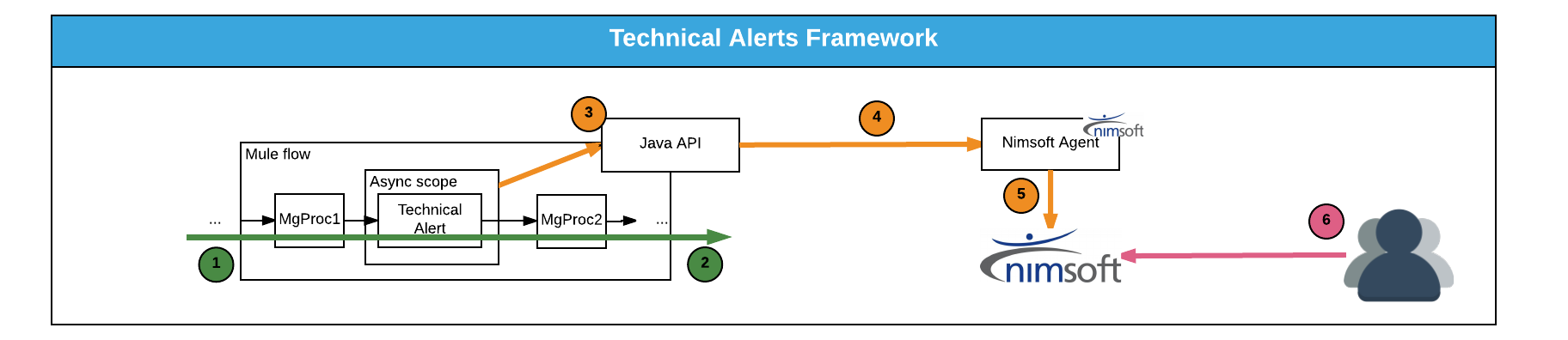
The fields that are available in the TO-DO list, and, therefore, used in the Business Alerts Framework are listed in the following table:

|  |  |  |
| --- | --- | --- |
| Field Name | Type | Restrictions |
| dateTimeTagFormat | String | Required.Example:  yyyy-MM-dd-hh.mm.ss |
| toDoEntryId | string | length=14 |
| toDoType | string | length=8 |
| toDoRole | string | length=10 |
| assignedTo | string | length=8 |
| sendTo | string | Possible values: SNDR , SNDU |
| entryStatus | string | Possible values: C , O , W (Default to ‘O’ in case of a TODO task) |
| createDateTime | dateTime | yyyy-MM-dd-hh.mm.ss |
| assignedDateTime | dateTime | yyyy-MM-dd-hh.mm.ss |
| assignedUser | string | length=8 |
| completeDateTime | dateTime | yyyy-MM-dd-hh.mm.ss |
| completeUser | string | length=8 |
| completeSW | boolean | default=false |
| comments | string | length=254 |
| priority | string | Possible values: 010, 020, 030, 040, 050, 060, 070, 080, 090 |
| batch | string | length=8 |
| batchNumber | decimal | From -9999999999 to 9999999999 |
| messageCategory | decimal | From -99999 to 99999 |
| messageNumber | decimal | From -99999 to 99999 |
| subject | string | length=30 |
| isNewCase | boolean | default=true |
| drillKey1 | string | length=50 |
| drillKey2 | string | length=50 |
| drillKey4 | string | length=50 |
| drillKey5 | string | length=50 |
| sortKey1 | string | length=2000 |
| sortKey2 | string | length=2000 |
| sortKey3 | string | length=2000 |
| sortKey4 | string | length=2000 |
| sortKey5 | string | length=2000 |
| messageParm1 | string | length=2000 |
| messageParm2 | string | length=2000 |
| messageParm3 | string | length=2000 |
| messageParm4 | string | length=2000 |
| messageParm5 | string | length=2000 |
| messageParm6 | string | length=2000 |
| messageParm7 | string | length=2000 |
| messageParm8 | string | length=2000 |
| messageParm9 | string | length=2000 |

#### 4.3.3.4. Technical Alerts Framework

As part of the notifications that will take place in case of technical errors, there has been decided that this “Alerts” should be directed to Nimsoft, as part of the system monitoring tool used by the operations team.

The framework will consist on a component (message processor) that will receive information injected from the Mule flow, and it will redirect that information to the Nimsoft Agent, via an API or system call (command line call).



1. The message enters in the mule flow
2. The message exits from the mule flow
3. During the execution of the mule flow, a technical alert component is placed, that sends the content of the alert to the framework
4. The framework sends the alert to the Nimsoft Agent, via Java API.
5. The Nimsoft agent sends the alert to the Nimsoft server.
6. An operator opens the console and see the alert.

The format of the alert (attributes) sent to Nimsoft is:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| severity | integer | The severities of the alarms are:  0 - clear  1 - information  2 - warning  3 - minor  4 - major  5 - critical |
| subsystem | String | The subsystem that is the object of the alarm. |
| message | String | A text message with the details of the alarm |

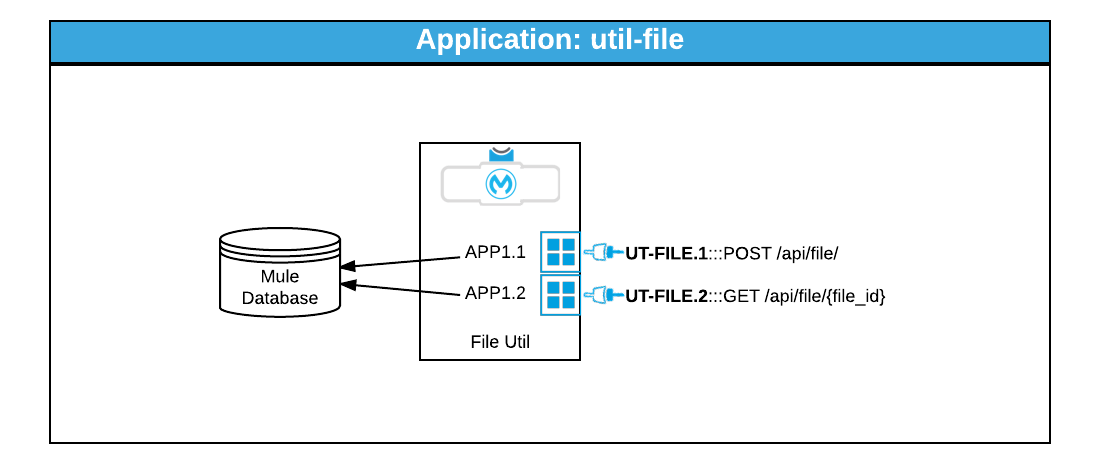
There are additional parameters accepted by the API, that are described here:

<http://docs.nimsoft.com/prodhelp/en_US/Monitor/SDK/Java/com/nimsoft/nimbus/NimAlarm.html>

The critical alarms (severity 5) will be treated with high priority by operations and invoke potential callouts out of business hours . <COMPANY NAME> will provide the details for the severity and priority of alerts across the business processes.

#### 4.3.3.5. APP1-File Util

This application will be part of the core of the solution, as it will be highly reused by the rest of the applications:



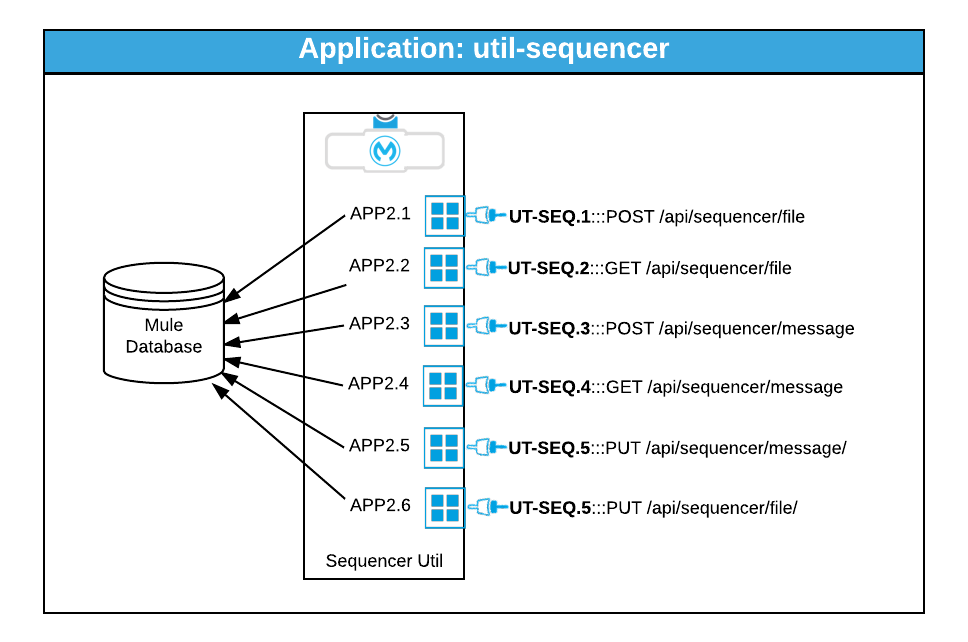
##### 4.3.3.5.1. APIs Exposed

|  |  |  |  |
| --- | --- | --- | --- |
| API Code | Method / Endpoint | Implementation Flow | Security Roles / Groups granted |
| **UT-FILE.1** | POST /api/file/ | APP1.1 | UT-FILE-RW |
| **UT-FILE.2** | GET /api/file/{file\_id} | APP1.2 | UT-FILE-RO  Or  FULL\_ACCESS |

##### 4.3.3.5.2. Functionality / Flows

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Flow Code | Name | Trigger and Pattern | Description | Endpoints/APIs called |
| APP1.1 | Insert file | API endpoint, Synchronous | Receives an API Call and insert a record in a DB table with file metadata and the file content. Returns the file\_id for future reference of the caller | INSERT statement in a DB Table |
| APP1.2 | Get file | API Call, Synchronous | Receives an API call with a file\_id and reads the record from the DB table with that file\_id, returning it in binary format. | SELECT statement in a DB Table, returning the binary content of the file. |

#### 4.3.3.6. APP2-Sequencer Util



##### 4.3.3.6.1. APIs Exposed

|  |  |  |  |
| --- | --- | --- | --- |
| API Code | Method / Endpoint | Implementation Flow | Security Roles / Groups granted |
| UT-SEQ.1 | POST /api/sequencer/file | APP2.1 | UT-SEQ-RW |
| UT-SEQ.2 | GET /api/sequencer/file | APP2.2 | UT-FILE-RW |
| UT-SEQ.3 | POST /api/sequencer/message | APP2.3 | UT-SEQ-RW |
| UT-SEQ.4 | GET /api/sequencer/message | APP2.4 | UT-SEQ-RW |
| UT-SEQ.5 | PUT /api/sequencer/message | APP2.5 | UT-SEQ-RW |
| UT-SEQ.6 | PUT /api/sequencer/file | APP2.6 | UT-SEQ-RW |

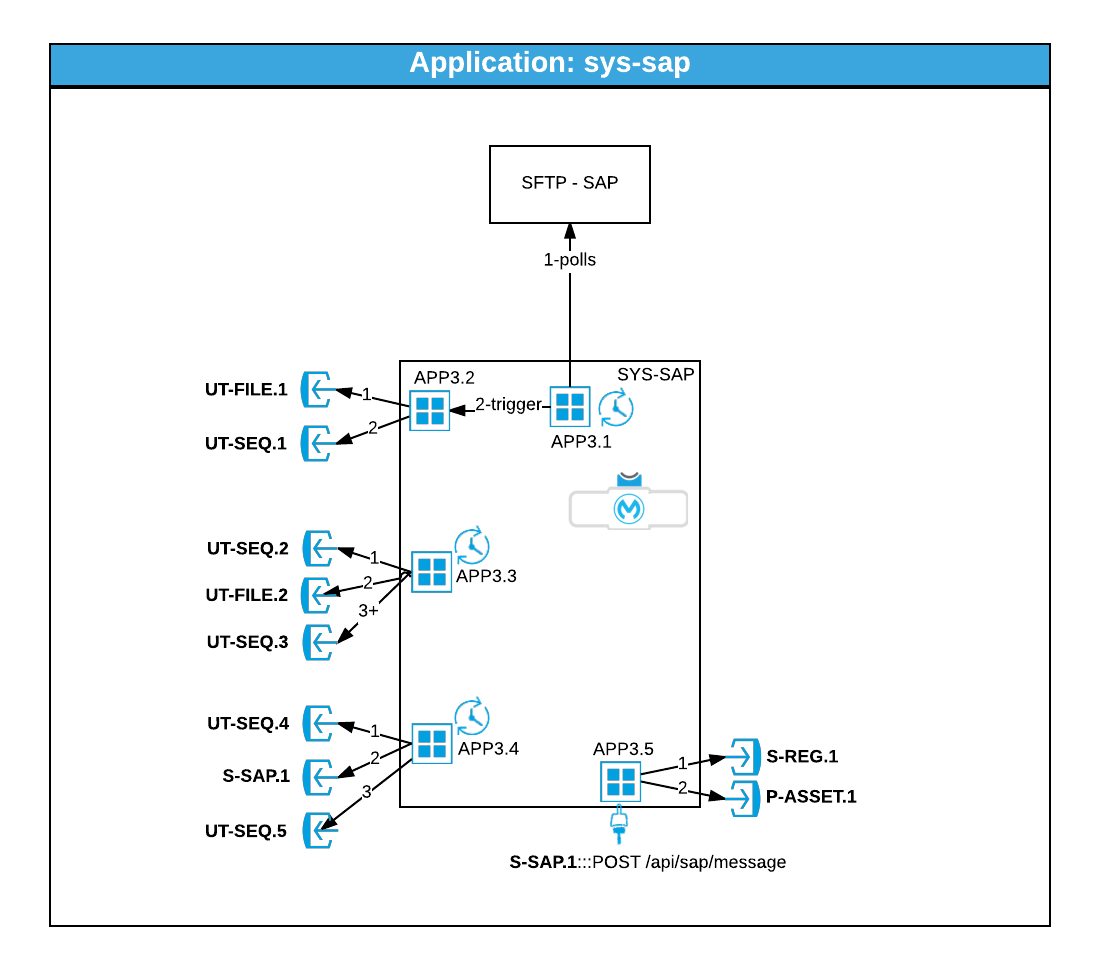
##### 4.3.3.6.2. Functionality / Flows

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Flow Code | Name | Trigger and Pattern | Description | Endpoints/APIs called |
| APP2.1 | Create a request to process file | API Call, synchronous | Inserts a request to process a file. It returns the file request id. | DB procedure or DB Insert |
| APP2.2 | Get and lock a request to process a file | API Call, synchronous | Returns a file request id and mark that file as “ON\_PROGRESS”. No other request can obtain the same request id back, and, therefore, the file processing task is assigned to the caller who got the request id (this locks the record). | DB procedure |
| APP2.3 | Create a request to process a message | API Call, synchronous | Inserts a request to process a message. It returns the message request id. | DB procedure or DB Insert |
| APP2.4 | Get and lock a request to process a message | API Call, synchronous | Returns a message request id and mark that message as “ON\_PROGRESS”. No other request can obtain the same message request id back, and, therefore, the message processing task is assigned to the caller who got the message request id (this locks the record). | DB procedure |
| APP2.5 | Notifies about the status result after processing a message | API Call, synchronous | Updates the status of a message request id after completing the processing of that message. It sets the status to COMPLETED or ERROR. Also, this functionality checks for the overall file status (in case the message belonged to a file) to mark that file as COMPLETED\_OK or COMPLETED\_WITH\_ERRORS. | DB procedure |

### 4.3.4. System APIs

The following sections describe the different applications that are included in the System APIs layer

#### 4.3.4.1. APP3-Sys-sap



##### 4.3.4.1.1. APIs Exposed

|  |  |  |  |
| --- | --- | --- | --- |
| API Code | Method / Endpoint | Implementation Flow | Security Roles / Groups granted |
| S-SAP.1 | POST /api/sap/message | APP3.5 | S-SAP-RW |

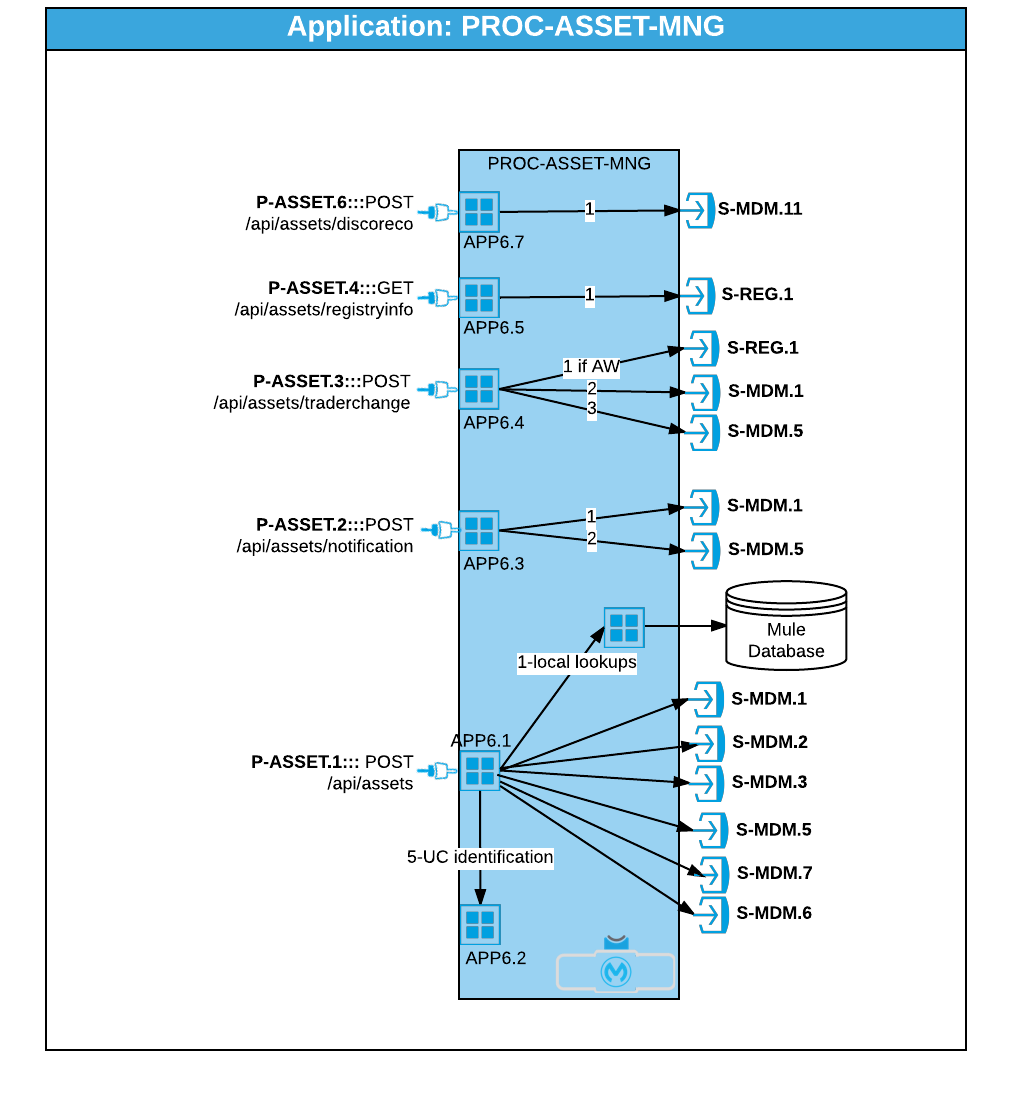
##### 4.3.4.1.2. Functionality / Flows

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Flow Code | Name | Trigger and Pattern | Description | Endpoints/APIs called |
| APP3.1 | Get File | Polling pattern. Asynchronous | This flow polls for the existence of files in the remote SAP SFTP server, and if there is one available, then triggers the flow APP3.2 |  |
| APP3.2 | Inserts file and request | Asynchronous, internal flow | This flow is triggered by the APP3.1 flow and it inserts the file in the file utility, and after it inserts a request to process the file using the Sequencing util | UT-FILE.1  UT-SEQ.1 |
| APP3.3 | Process File | Poll, Asynchronous | This flow request for an available file to be processed. It gets the file, opens it and for every message it store the message in the sequencer table | UT-SEQ.2  UT-FILE.2  UT-SEQ.3 |
| APP3.4 | Get Message | Poll, Asynchronous | This flow request for an available message to be processed. It gets the message, process it by calling the external API S-SAP.1, and it reports back the status of the processing | UT-SEQ.4  S-SAP.1  UT-SEQ.5 |
| APP3.5 | Process Message | API, Synchronous | This flow receives an inbound message, enrich it with information from the Registry, transform to canonical and calls the process api in a canonical format | S-REG.1  P-ASSET.1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| APP7.12 | Create TODO | API Synchronous | Transform from Json to SOAP and response back to Json | SOAP Web service in MDM |
| APP7.14 | Create Disco/Reco Event | API Synchronous | Transform from Json to SOAP and response back to Json | SOAP Web service in MDM |

### 4.3.5. Process APIs & Orchestration Applications

#### 4.3.5.1. APP6-Proc-assets-mng



##### 4.3.5.1.1. APIs Exposed

|  |  |  |  |
| --- | --- | --- | --- |
| API Code | Method / Endpoint | Implementation Flow | Security Roles / Groups granted |
| P-ASSET.1 | POST /api/assets | APP6.1 | P-ASSET-RW |
| P-ASSET.2 | POST /api/assets/notification | APP6.3 | P-ASSET-RW |
| P-ASSET.3 | POST /api/assets/traderchange | APP6.4 | P-ASSET-RW |
| P-ASSET.4 | GET /api/assets/registryinfo | APP6.5 | P-ASSET-RW  P-ASSET-RO |
| P-ASSET.6 | POST /api/assets/discoreco | APP6.7 | P-ASSET-RW |

##### 4.3.5.1.2. Functionality / Flows

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Flow Code | Name | Trigger and Pattern | Description | Endpoints/APIs called |
| APP6.1 | Process Assets | API, Synchronous | Processes information about a Service Point, Installation Event and Device. It orchestrates lookups into MDM to find out the status, as well as invoking the APP6.2 to find out the Use Case. In the end stage, it invokes the sync SP, IE and/ or Device in MDM based on the use case identified. | S-MDM.1  S-MDM.2  S-MDM.3  S-MDM.5  S-MDM.6  S-MDM.7 |
| APP6.2 | Use Case Resolver | Internal Call (flow), Synchronous | It resolves the Business Use Case depending on the input message, as well as the status in MDM. | none |
| APP6.3 | Notifications Processor | API, Synchronous | It processes a notification coming from the Registry, that updates 1 or more Service Points with Network and ANZIC code | S-MDM.1  S-MDM.5 |
| APP6.4 | Trader Change Processor | API, Synchronous | It processes a change in the Trader for a particular Service Point. It finds the new trader (in case of AW cases) by calling the registry) | S-MDM.1  S-MDM.5  S-REG.1 |
| APP6.5 | Get registry info | API,  Synchronous | It gets information from the Registry regarding a particular Service Point. | S-REG.1 |
| APP6.7 | Submit Disco/Reco | API,  Synchronous | It submits the Disco/Reco event into the MDM Process API for disco/reco | S-MDM.11 |

### 4.3.6. Experience APIs

At the moment of this analysis, there hasn’t been identified any Experience API’s.

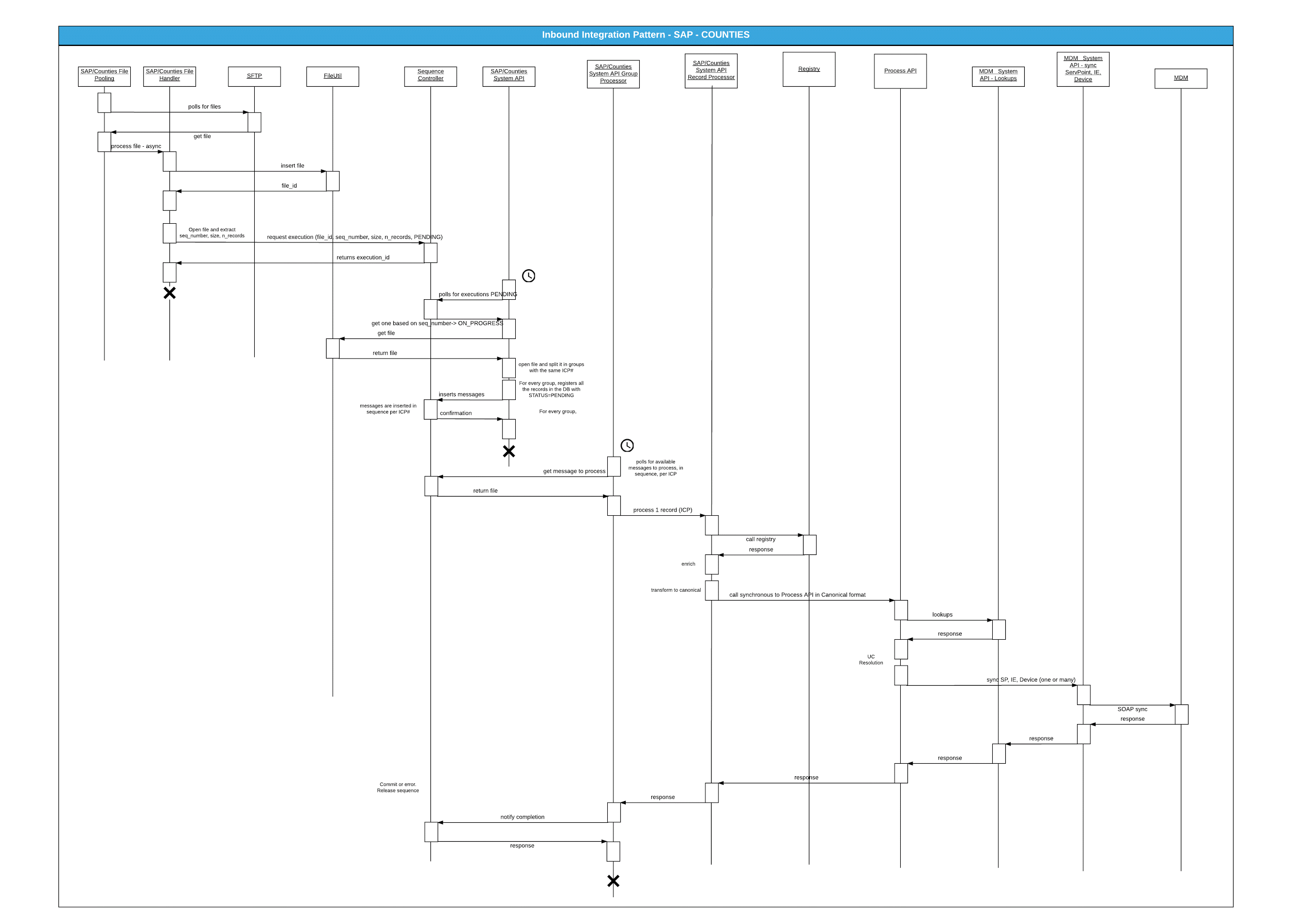
## 4.4. Application Interaction and Functionality View

The information seen before about the applications is a static view of them. In this section, we are going to see how these applications interact each other to achieve the outcome described in the business process.

Therefore, the following sequence diagrams show how the information is flowing from systems and mule and to systems.

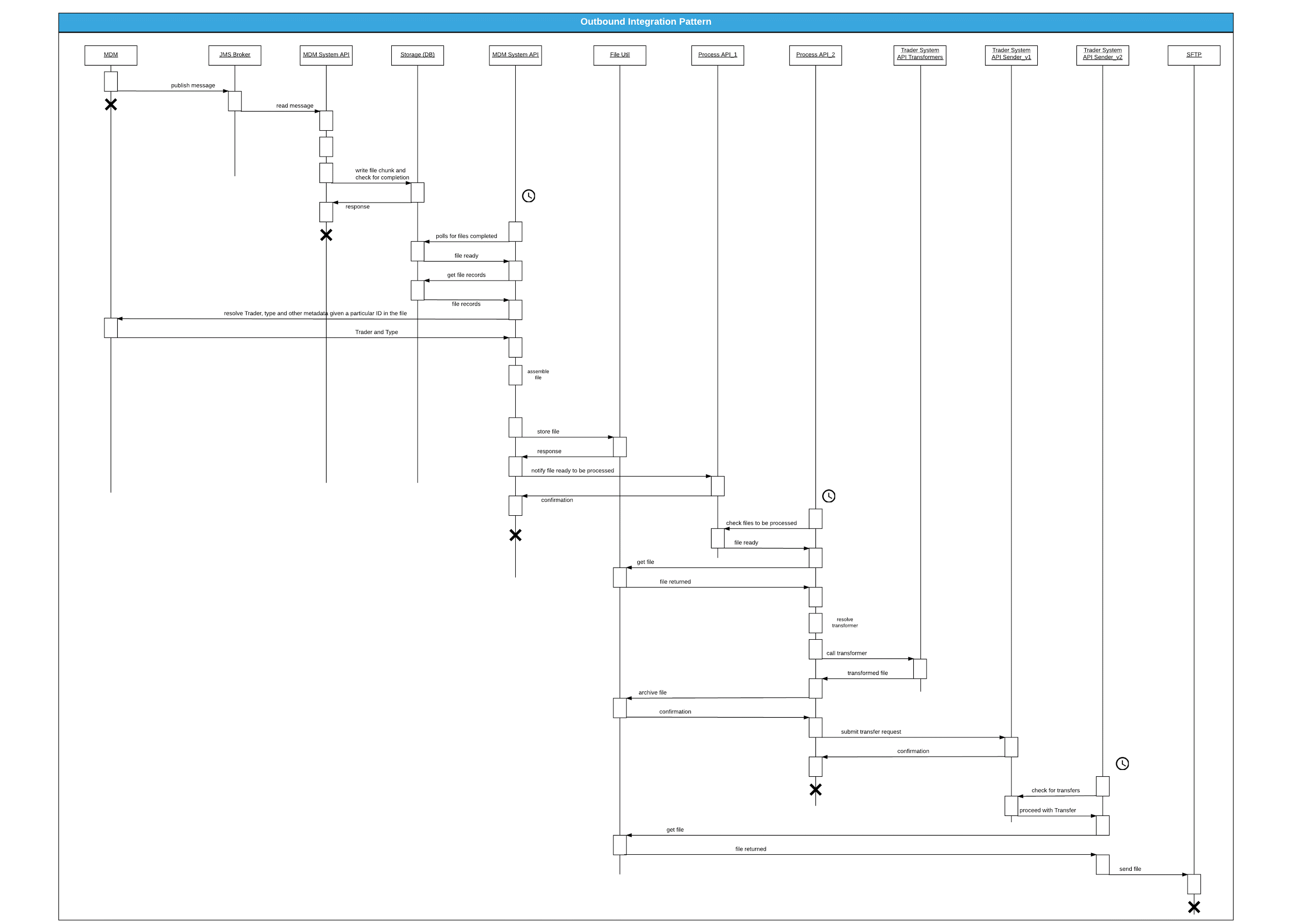
### 4.4.1. Inbound Interactions - SAP/COUNTIES

This process describes the interactions between the applications that take part when SAP and COUNTIES sends files with the changes in their systems.



### 4.4.4. Outbound Interactions - Consumption Report

This diagram describes the interactions taking place when a consumption report is executed in MDM, and how the report pieces are read by Mule, consolidated in one single file, transformed and transferred to the relevant Trader.

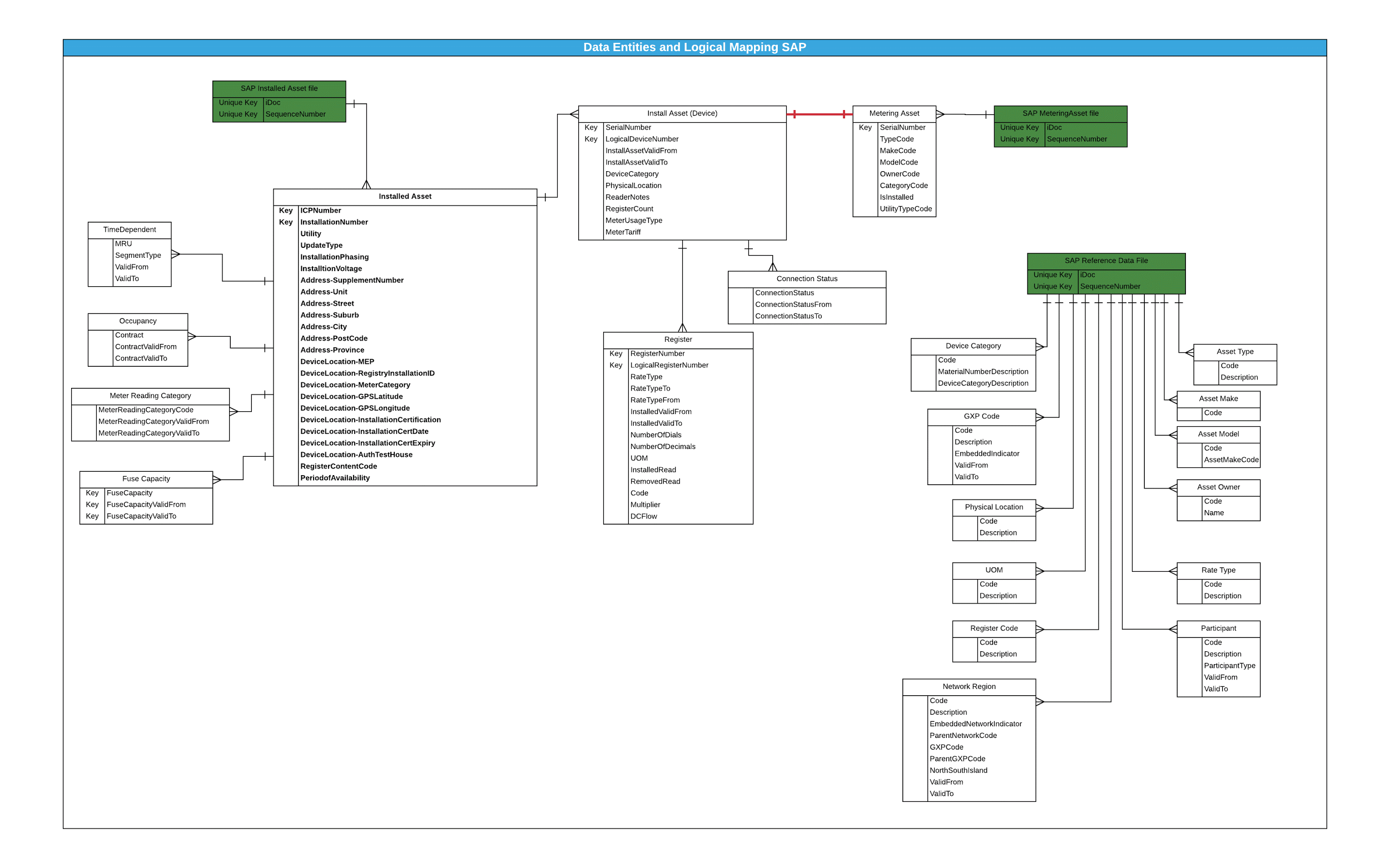


## 4.5. Data and Information View

In this section, a brief description of the data payloads of the systems that participate in the integration is provided, as well as the entity mapping rules (Unique Keys and Entity Mapping) are established in order to uniquely identify records in the source and target systems without ambiguity.

### 4.5.1. SAP Data model

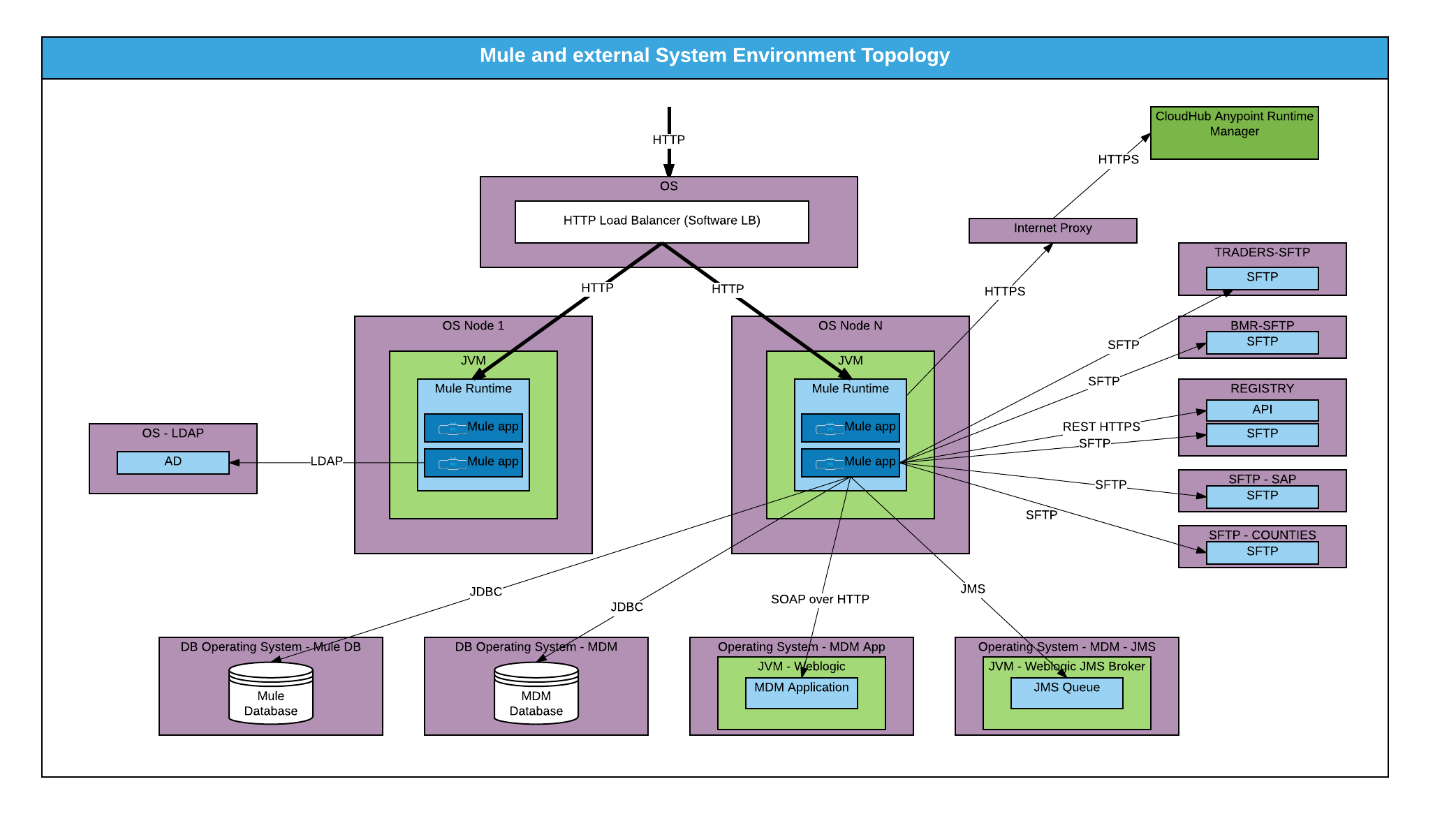
The information coming from SAP is reflected in this diagram:



## 4.6. Deployment and Physical View

The following diagram shows the generic Mule topology and the rest of external system that are going to interact with it.

***NOTE: In this diagram, any Mule application deployed in any Node will have connectivity with all the external systems. The diagram does not reflect that situation for aesthetic and clarity reasons (to avoid crossing lines).***



A cluster is a set of Mule instances that acts as a unit. In other words, a cluster is a virtual server composed of multiple nodes. The servers in a cluster communicate and share information through a distributed shared memory grid. That means data is replicated across memory in different physical machines. One can deploy, monitor or stop all the nodes in a cluster as if they were a single Mule server.

**The benefits of Clustering**

Clustering ensures high system availability. If a node becomes unavailable due to failure or planned downtime, another node in the cluster can assume the workload and continue to process existing events and messages.

Because all nodes in a cluster process messages simultaneously, clusters can also improve performance and scalability. Compared to a single node instance, clusters can support more users or improve application performance by sharing the workload across multiple nodes or by adding nodes to the cluster.

Beyond benefits such as high availability through automatic failover, improved performance, and enhanced scalability, clustering Mule instances offer the following benefits:

· Automatic coordination of access to resources such as files, databases, and FTP sources. Mule automatically manages which node will handle communication from a data source.

· Automatic load balancing of processing within a cluster. If you divide your flows into a series of steps and connect these steps with a transport such as VM, each step is put in a queue, making it cluster enabled. Mule can then process each step in any node, and so better balance the load across nodes. This load balancing shouldn’t be confused with the Load Balancing that distributes the load of the HTTP traffic, outside of Mule.

· Polling shared resources (scheduler, shared object stores) through one common objects repository.

· Cluster lifecycle management and control.

· Cluster and node performance monitoring.

· Raised alerts. You can set up an alert to appear when a node goes down and when a node comes back up.

### 4.6.1. Environment: TEST1

This environment will be used for the performance testing purposes. The details of that environment will be provided in a separate document as part of the detail environment design. These details required can be seen in the Appendix of this document.

### 4.6.2. Environment: TEST2

This environment will be used for end to end functional testing . It will have half of the data-load from the production environment. This environment is not HA, it is just a single 2 core server.

Note: Similar configuration tables are required to specify the TST2 systems and environments. They will be added once they are ready.

### 4.6.3. Environment: PRE-PROD (UAT)

This environment will be set up for User Acceptance Testing, BAU and Application support team .

### 4.6.4. Environment: PROD

This is the environment used for production purposes

### 4.6.5. Environment: DR

The DR pattern to be followed for <COMPANY NAME> would be Active -Passive . Hence a cold DR environment will be set up. The DR environment will be provisioned by the existing DR solution which is snap mirroring across the secondary site. So there is no expectation DR will be directly delivered (It will be provisioned for by the existing DR solution in place for prod).

## 4.7. Software Versions

TODO-04: Software Versions. There are still some data to be completed once the infra design is done by <COMPANY NAME>.

|  |  |  |
| --- | --- | --- |
| Element | Name | Version |
| Mule Version | Mule Runtime | 3.8.1 |
| Java | Oracle Java Development Kit | Version 8 update 111 |
| Maven | Apache Maven | 3.3.9 |
| Operating System | ?? | ?? |
| Database | ?? | ?? |
| SFTP | ?? | ?? |
| Load Balancer | ?? | ?? |
| Active Directory | ?? | ?? |

## 4.8. Transports and Protocols View

The following table describes the mechanisms that the information is going to flow from one system to another, from the Transport protocol perspective.

|  |  |  |
| --- | --- | --- |
| System | Transport | Environments |
| SAP-FTP | SFTP (FTP over SSH) | DEV: Yes  TEST1: Yes  TEST2: Yes  PREPROD:Yes  PROD:Yes |
| MDM-Database | JDBC over TCP | DEV: Yes  TEST1: Yes  TEST2: Yes  PREPROD:Yes  PROD:Yes |
| MDM-WebServices | SOAP over HTTPS | DEV: Yes  TEST1: Yes  TEST2: Yes  PREPROD:Yes  PROD:Yes |
| MDM-JMS | JMS | DEV: Yes  TEST1: Yes  TEST2: Yes  PREPROD:Yes  PROD:Yes |
| Active Directory | LDAP | 1 or all the environments |
| CloudHub Anypoint Runtime Manager | HTTPS | 1 for all the environments |

## 4.9. Security View

### 4.9.1. Inbound Security Enforcement

#### 4.9.1.1. Authentication

The API is available privately over https, but requires authentication via a username and password.

All the system users will be available in Active Directory under the relevant roles to access the backend APIs.

#### 4.9.1.2. Authorization

The roles and groups will be defined in the centralised repository which is Active Directory in case of <COMPANY NAME>. Role based access control will be managed via Active Directory

The implementation of Authentication and Authorisation will be done via the standard Spring Security framework .Mule allows to have several Security Providers as part of Mule Security Manager context definition. Each Security Manager is associated with an Authentication Provider using the delegate-ref attribute. In a nutshell, an Authentication Provider is a wrapper for providing authentication services. For example, org.springframework.security.authentication.dao.DaoAuthenticationProvider is the class which retrieves user details from a simple UserDetails Service (often backed by a properties file containing users and groups ).

All these security-related objects exist in a hierarchy as follows:

Mule Security Manager -> Spring Security Provider -> Ldap Authentication Provider -> Ldap Server Definition

* **Configure the Spring Security Manager**

<!-- Mule's Spring Security Manager -->

<mule-ss:security-manager>

<mule-ss:delegate-security-provider name="SpringSecurityProvider" delegate-ref="SpringAuthManager" />

<mule-ss:delegate-security-provider name="SpringSecurityProviderUT" delegate-ref="SpringAuthManagerUT" />

</mule-ss:security-manager>

* **Configure the authentication Manager to point to AD**

<!-- Spring Authentication Manager -->  
<ss:authentication-manager alias="SpringAuthManager">  
<ss:ldap-authentication-provider   
 server-ref="LdapServer"   
 user-search-filter="(uid={0})"   
 user-search-base="ou=People,dc=YourOrg,dc=com,dc=au"   
 group-search-base="ou=Accessgroups,dc=YourOrg,dc=com,dc=au"   
 group-search-filter="(memberUid={0})"   
 group-role-attribute="cn"  
 />   
</ss:authentication-manager>

* **LDAP as a Spring Security Provider**

<ss:ldap-server id="LdapServer" url="ldap://ldap.yourorg.com.au:389"/> //ad url

One of the considerations is to cache the credentials after the LDAP query . This can be implemented within the Spring Framework.

The implementation required is to cater for the following steps :

* User details are retrieved and UserCache is used to retrieve previously cached user details (i.e. getUserFromCache is called either by implementations of UserDetailsService or AuthenticationProvider before the call to AuthenticationManager is performed).
* After basic pre-authentication checks (credentials expiration etc.) the actual authentication occurs. At this point the password from cached user details is compared to the password stored in Authentication object supplied.

#### 4.9.1.3. Encryption (SSL)

At present there are no defined requirements for SSL implementation as all the services will be within the same VLAN.

### 4.9.2. Outbound Security Configuration

The transport level security will implemented via the following :

SFTP - The files to be shared with the traders will be done via SFTP using username/password.Username and password for SFTP server connection will not be LDAP integrated, as the server is currently not integrated with the <COMPANY NAME>metering domain. i.e. the account on the SFTP server will be a local account setup for access

APIs - The APIs will be protected with basic authentication . Currently there is no API Policy implementation for security as all APIs are private/internal and will not be exposed via a public interface .

Registry API - The API will be accessed via a key provided by Registry

JDBC- The database calls will be made via JDBC with appropriate credentials . The credentials would be applicable for the CRUD operations required to be performed .

JMS - Currently there is no security implemented on Active MQ JMS interface .Moving forward , the JMS interactions via Weblogic JMS will be secured via a security policy which could be credentials based (security principle ,security credentials) .

SOAP over HTTP - The webservices will be secured by basic HTTP authentication.

## 4.10. Persistence Store View

In the current implementation, SQL persistence store is used for the following purposes:

1. Storage of the incoming files
2. Storage of the outbound files
3. Storage of the messages
4. Configuration or Property files

This will provide capability to track and replay the messages if required .

## 4.11. Error Handling View

There are two different types of errors that can occur in Mule:

·System Exceptions - This is when no message is involved, for instance an error on application startup

·Messaging Exceptions - This is when there is an error within a flow

Exception-reporting comprising of messaging and system exceptions must be, at a minimum, logged to file, and optionally, depending customer preference, sent to an alerting system.In the event that all flows adopt the same exception strategy, it is a best practice to define the exception flow at the global level and ensure that all other flows references the global strategy via the use of a reference exception strategy.

For the current design , all the messaging exceptions /business exceptions will be catered in MDM . Mule will create a TODO task in MDM via existing API and these exceptions will be manually resolved within MDM. At this point , the message will be marked as in error and further messages related to that ICP will not be processed until the TODO task is resolved in MDM.

In case of Technical exceptions , Mule will create an alert on the agent using ?? API which will be passed on to ?? server ‘to be resolved by the Devops.

# 5. Implementation Guidelines

## 5.1. Logging and Auditing

### 5.1.1. Definitions

* **Logging**: A log is informational in nature - tracing execution, transformations, errors, notifications etc.; as useful and important as general logging may be it is typically not adequate to meet regulatory and security standards unless specific information is logged, and certain part of information are obscured or encrypted.
* **Auditing**: Audit-records are intended to be preserved for a defined period at minimum, and may be retained longer.

There are some key aspects that differentiate auditing from logging:

1. specific information is required to be present in audit-logs

2. certain fields, if they appear in audit logs at all, are required to be masked or encrypted.

3. The length of retention of audit-records is often regulated

4. Auditing is often related with a business event that has to be captured and retained for different purposes: Security, KPI, Monitoring, Root Cause Error, etc.

For logging, Mule uses slf4j, which is a logging facade that discovers and uses a logging strategy from the Java Classpath, such as Log4j2 or the JDK Logger. By default, Mule includes log4j2, which is configured via a file called log4j2.xml, located in src/main/resources.

### 5.1.2. Key Considerations and Best Practices applicable to Logging and Auditing

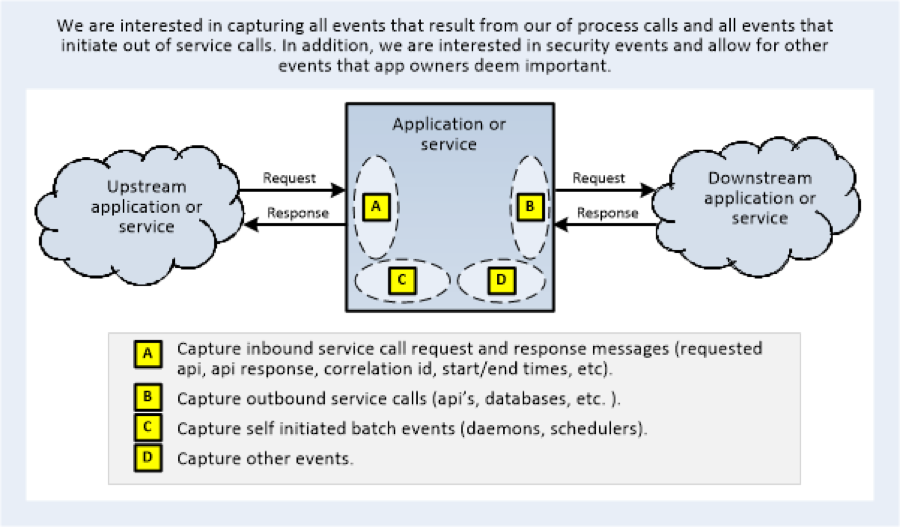
Some of the key considerations for logging in the Mule application:

* Apply caution when logging parts of messages, especially the payload. Logging messages can bypass streaming and incur significant disk IO for large payloads. Payload logging should only be used with DEBUG level logging.
* Make sure you don’t close the stream for logging purposes.
* Use log levels to identify DEBUG, INFO, WARN and FATAL logs.
* The Mule Runtime uses the asynchronous log4j2 logging framework. Log4j2 tends to be very efficient in contrast to its synchronous predecessor.
* It is possible to have multiple logging styles, you can even log errors synchronously and the debug level asynchronously, providing easy access to problems. But keeping performance in mind, logging should always be asynchronous.
* Consider developing a purpose built Mule Logger, using DevKit. The benefit of using a common logger is that messages can be standardised, common information particular to a customer can be auto-logged (e.g. application specific payloads), etc. MuleSoft Processional Service can provide a Proof of concept of a Logging connector which can be tailored for <COMPANY NAME> requirements.

The events that should be allowed for in the Enterprise Logging are: Inbound, Outbound, Batch and Other. These are detailed below:

* Inbound - represents all requests that are made to the logging application/service/appliance.
* Outbound - represents all requests that are made by the logging application to other applications/services/appliances.
* Batch – represents self-initiated requests like daemons and internal schedulers as is often the case in batch processing.

Other - represents a catch all for any other events the logging application wishes to log. This includes security events specified in the Application Security Logging Guideline.



The above diagram depicts requests as they come into a system (this applies to queue, or OLTP systems). The ‘A’ oval represents the scope of an Inbound request and the corresponding event, the ‘B’ oval an outbound request to an external system and the corresponding event, the ‘C’ oval is for self-initiated batch type internal requests such as kicked off by daemons or scheduling routines, and the ‘D’ oval any other logged event.

The event type is specified as a portion of each and every log record as a part of the Header Field. The log events must contain sufficient content to enable analysis of the platform’s behavior from both internal and external perspective. Details include items of technical significance as well as business significance. This content makes up the Fields of the log record.

The Log record Fields are categorised into standard Contexts as below:

1. Header – standard identifiers of the log event - application id, event type, timestamp
2. Source Context – contains data about the origination of the event – the user/device/carrier/client application/business context.
3. Application Context – contains data about the processing platform and application where the event originated and is currently being processed
4. Service Context – this is the “what” that is being invoked. It contains technical details of the call (will vary according to event type) – service endpoint, service key etc.,
5. Actor Context – The IDs of the actors invoking the call.
6. Security Context – Security specific information. The fields required here will be defined in the Application Security Logging Guideline.
7. Service Results – timings, data size, error codes/messages.

Along with effective logging, it is important to achieve end-to-end traceability to ensure inter and intra system transactions are effectively captured for the fault diagnosis.

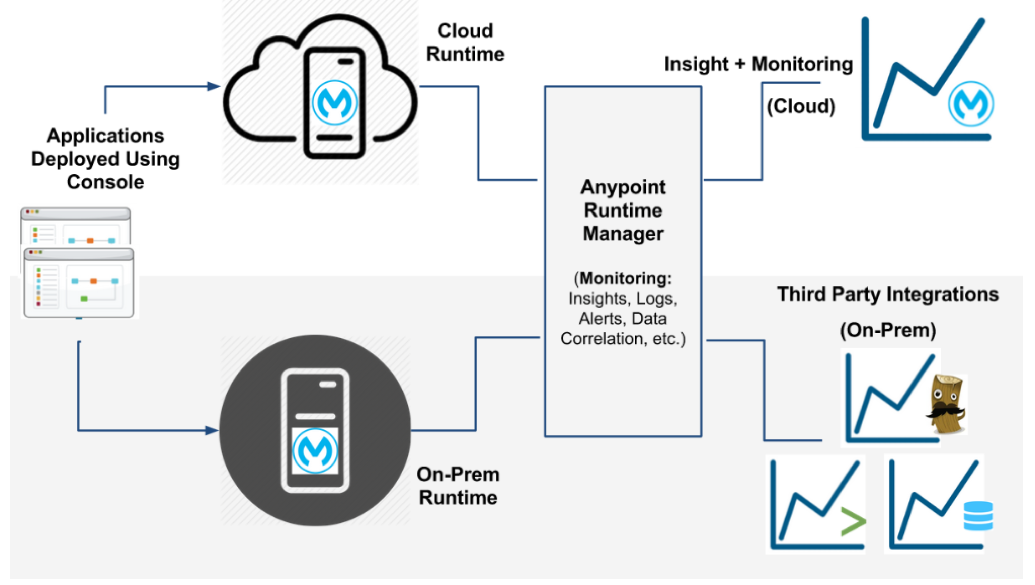
For application the traceability should be done in two ways.

* End to end traceability → An audit id should be used for logging the whole end to end process . This helps in providing information of the whole transaction.
* System to system calls → If the current flow is interacting with another system as a part of orchestration ,then a transaction id or correlation id should be defined to identify the outcome of that system call.

Both end to end audit id and correlation id per call help in providing effective traceability of the application.

Enterprise Logging can be implemented in the following ways:

* Custom connector for Common services: Develop a custom connector to provide logging services using Any Point Connector Devkit. The log4j configuration can be modified to persist information to file system or database depending upon <COMPANY NAME> requirements.
* Sending Data from Runtime Manager to External Monitoring Software (this configuration is only currently available for on premise Mule Servers



Currently for on-premise Mule Servers, MuleSoft enables users to seamlessly integrate into their Splunk/ELK instances. This easy configuration will be available for CloudHub Workers in the future.

Auditing Data fields:

Details about the auditing fields are provided in the section Auditing Framework . It is a component (message processor) that will write messages into a audit database table with a particular structure as finalised above .

## 5.2. Naming Conventions

### 5.2.1. Artefacts and Components naming conventions

|  |  |
| --- | --- |
| Mule Configuration XML Naming Convention | Description |
| Main-   * main-service * main-queue * main-rest * main-scheduler * main-poller * main-file * main-api | All entry-level flow configuration files should be prefixed with main-. Entry level flow configuration files are all flows that expose one or multiple endpoints. Examples of these endpoints include web services, queue receivers, restful services and etc. It is a good practice to dedicate each flow to a particular endpoint type. For example if a project exposes both queue receivers and web services then place the queue receivers in their own dedicated Mule Configuration XML file and the web services in their own XML as well. The text that follows the prefix is a short description of the endpoint type and some examples have been provided on the left hand column and include:   * main-service: Web Service entry points * main-queue: Queue Receivers * main-rest: Restful Services * main-scheduler: Scheduled flows * main-poller: Polled Data Sources * main-file: File Pollers * main-api: APIKit exposed services   These main level flows can contain sub-flows but it must be ensured that sub-flows associated with these main flows are in fact only dedicated to these flows. If they are sub-flows that can be used by other flows then it would be a better idea to place them in their own Mule configuration XML file. Sub-flows ensure that the provided visual implementation is more readable and modularized to facilitate reuse and maintenance. The remainder of the flow name should be based on the business context that the flow is addressing. |

|  |  |
| --- | --- |
| Flow Naming Convention | Description |
| main:   * main-api:service | All entry-level flows should be prefixed with the name of the Flow Configuration File Name. Entry-level flows are all flows that expose one or multiple endpoints. Examples of these endpoints include web services, queue receivers, restful services and etc. It is a good practice to dedicate each flow to a particular endpoint type. For example if a project exposes both queue receivers and web services then place the queue receivers in their own dedicated XML and the web services in their own XML as well. The text that follows the prefix is a short description of the endpoint type and some examples have been provided on the left hand column and include:   * main-service:ws\_entry: Web Service entry points   These main level flows can contain sub-flows but it must be ensured that sub-flows associated with these main flows are in fact only dedicated to these flows. If they are sub-flows that can be used by other flows then it would be a better idea to place them in the shared\_ section. Sub-flows ensure that the provided visual implementation is more readable and modularized to facilitate reuse and maintenance. The remainder of the flow name should be based on the business context that the flow is addressing.  For projects using API Kit Router, generated flows based on API resources & operations must not be renamed. |
| sf-   * sf-ops- * sf-api- * sf-q- * sf-f- | All sub-flows should be prefixed with sf-. Each sub-flow provides a specific type of processing and as such the short description that follows the initial prefix must address this type of processing. For example ‘ops’ to highlight the implementation of a web service operation, ‘q’ to address the processing of a JMS queue and others. As per the flow names, the text provided after these prefixes must be related to the business context that the sub-flow is addressing. |
| Shared-   * globals- * flows- * sf- | All components that are common across the project should be placed in an XML file that is prefixed by shared-. In the event of XML files dedicated to shared resources such as Beans, Connectors, other, it is perfectly OK for the XML to NOT contain any message flows. The short description following the initial prefix is related to the type of common resources e.g.   * globals-: connectors, beans, global strategies * flows-: common flows across project * sf-: common sub-flows across the project   The remainder of the file name is an optional text that should once again be related to the business context. |

MuleSoft recommends the following naming conventions:

·Application Name

oLowercase

oDasherized

oe.g. track-and-trace-api

·Mule Configuration XML

oLowercase

oDasherized

oEntity, system or functional grouping

oe.g. user-notifications.xml, user-roles.xml

·Flow Name

oPrefixed with Mule Configuration XML name and colon forward slash

oLowercase

oDasherized

oe.g. user-roles:/get-role-by-id

·Inbound / Outbound Endpoints

oReadable description of the endpoint and function

oe.g. Invoke Elastic Search Service OR

oProtocol plus URI, queue name

oe.g. TCP:/my-uri, JMS:/my-topic-name

·Flow Reference Step Name (doc:name)

oDirect copy of the referenced flow name using the naming above

·Flow Step Name (doc:name)

oTitle case with spaces

oReadable description of the step function including choice routers

oThe default component name should never be used

oe.g. Log Message ID, Check User Role etc.

·Flow Variables

oCamel case

oMeaningful name defining use or function

oMetadata defined where possible

oe.g. isAdmin, customerId

·Property Name

oLowercase

oDot separators

oGrouped by system, protocol, function

oe.g. splunk.transactional.tcp.host

### 5.2.2. Deployable Mule Application Archive Naming Convention

Mulesoft advise on using Apache Maven to build Mule Applications. Apache Maven support is built into Anypoint Studio. By only defining a Group Id and Artifact Name, deployed archives names are standardised and incorporates application name, versioning, and snapshot builds.

Provide a Group and Artifact Id, and Anypoint Studio will generate a Mule Maven Project, set the Classpath, and generate an Apache POM file which contains build related information, Mule & other dependencies, etc.

### 5.2.3. Source Code Best Practices

Mule's projects can be checked into any standard source control system, including CVS, SVN, Mercurial, Git and TFS among others.Not all files in a Mule project need to be checked into source control system. Some files are metadata generated by Studio and other tools in order to keep track of the project, others are temporary information generated by the build process and are completely machine-dependent. Checking in these files will cause conflicts in other team members' projects, or will cause their builds to fail.

Check in these files and folders:

· src/ folder (source code).

· the mule-project.xml file (the Anypoint Studio project descriptor).

· pom.xml (for Maven-based projects, the project's build configuration).

Do NOT check in these files and folders and add these in the Bitbucket ignore list (.gitignore). They are either temporary or:

· .classpath (Studio metadata)

· .settings (Studio metadata)

· target/ folder (contains compiled code)

· bin/ folder (contains compiled code)

· catalog (contains Mule project metadata)

Refer to the following link for more information on preparing a gitignore file:<https://docs.mulesoft.com/mule-user-guide/v/3.8/preparing-a-gitignore-file>

In general, the preferred workflow is:

· Create a branch from master, check it out, do your work

· Test and commit your changes

· Optionally push your branch up to the remote repository (origin) OR

· Optionally rebase your branch to master (if your changes are unpublished)

· Checkout master, make sure it's up-to-date with upstream changes

· Merge your branch into master

· Test again (and again)

· Push your local copy of master up to the remote repository master (origin/master)

· Delete your branch (and remotely, too, if you published it)

Some key points:

· Development in a local branch (even with occasional merges with master) is a valid and recommended development pattern. If parallel commits have been pushed to master in the meantime, this workflow will represent your local changes as if it indeed were a separate branch.

· Development in an unpublished local branch, and using GIT rebase instead of pull or merge to update the local branch with changes to master is also a valid pattern.

### 5.2.4. Continuous Integration

Setting up Continuous Integration for a Mule project is highly recommended. By using Apache Maven as the primary build dependency tool, provides ability to set-up a build that is triggered on every project change and execute all unit and functional tests automatically.

There are plenty of Continuous Integration tools available. For example Jenkins, TeamCity and Bamboo are popular choices.

In order to achieve Continuous Integration, the following steps are provided:

1. Mule Developer will push code changes into the source code repository, e.g. BitBucket or GitHub.
2. Either automatically (polling) or manually (via manual deployment) the Continuous Integration Server will pull code from BitBucket, execute a Mule Project build using Apache Maven, pull dependencies from configured Maven Repositories, compile source code and create a versioned Mule Application Deployable Archive file
3. The Continuous Integration Server will use the Mule Maven plugin to push the Mule Application Deployable Archive to CloudHub. The following image is an example of the configuration required within the Apache Maven POM file. Note that values are hardcoded in the example, but should be externalised in the Continuous Integration & Delivery tool, and parameters passed into the POM during the deployment task.

# 6. Mule Apps Configuration and Migration

How the configuration is going to be managed?

* Embedded in the applications with a file, one file per environment?
* Configured in a centralised database per environment?

There are two types of runtime properties that may be associated with an application and these are of relevance when an application is being migrated (deployed) from one environment to another.  These include properties that are environment agnostic and others that are environment specific.  It is recommended to separate these properties into separate properties files. Properties files are part of the application archive.  It is also recommended to include the environment name as part of the environment specific properties file in the form of a system level property. In the event that certain property values require encryption the **Secure Property Placeholder** will be used. Please note that this is placeholder is not part of the OOTB studio and must be installed within Anypoint Studio.  This placeholder is part of the **Anypoint Enterprise Security Module**. The encryption of properties is carried out within Anypoint Studio.

For the current implementation , the configuration properties will be persisted in the database . The columns containing confidential information such as passwords will be encrypted . The credentials to access the database will be encrypted using Mule Enterprise Security Property Placeholder.

For operational considerations , existing Stored procedures and scripts will be reused to facilitate the update to the property configuration including encrypted properties .

# 7. Architecture Registers

## 7.1. Decisions

### 7.1.1. About Error Handling Notifications

It has been decided that there will be 2 notification channels regarding when an exception happens:

* If it is a technical error (connectivity, wrong credentials, etc..), the alert will be sent to Nimsoft.
* If it is a business error (data validation error, use case data exception, etc..), the alert will be sent to MDM in the form of a TO-DO list.

### 7.1.2. About the usage of Database instead of JMS

It has been decided the usage of Database as a persistence store instead of the usage of JMS queues, due to the following requirements:

At file level:

* Requirement of sequencing the processing of files in order, following specific criteria.
* Different types of files will require different sequencing pipes.
* Different types of files will require different sequencing requirements (for example, there might be files that won’t require any sequencing).
* The ability to resubmit an already consumed file.
* The ability to change the sequence number by an operator, in case there is one file missing that is stopping following files to be processed.

At message level:

* Requirement of sequencing the processing of messages in order, following specific criteria, that depends on the message content (for example, sequence all the messages for the same ICP#, or for the same combination of ICP# and Serial#, or for the same Serial#).
* Depending of the message type, there might not be any sequencing requirement (so messages can be processed in parallel.
* Requirement of stop processing messages in one pipe if the current message gets in ERROR. The processing will resume after retrying of the failed message results in COMPLETE.
* Requirement to skip processing messages in one pipe.
* Requirement to be able to set manually (by an operator), a message to be SKIPPED.
* Requirement to be able to set manually (by an operator), a message to be RETRIED.

### 7.1.3. About the decision of using realtime web services vs asynchronous for the Inbound pattern.

There were two options to inject data into MDM:

* A) Using asynchronous batch processing pattern (JMS queues, staging tables + batch process)
* B) Using synchronous request-response pattern (web services).

The option chosen for the integration is B), because:

* MDM is going to be the system of reference about the asset information (service point, installation event, Devices), therefore, decisions about if a ServicePoint exists or not in MDM should be taken in realtime by the business logic in the context of a transaction.
* This requires that in the context of 1 transaction when processing an inbound message, there is a pre-step required to check what is the state in MDM (query to MDM), and, after obtaining that information, decide what to do with the inbound message (decisions about creating or updating a particular entity), which is the Use Case identification step.

If option A) were selected:

* The information obtained in the first step (query to MDM) cannot be trusted, due to “in flight transactions” happening because of the batch pattern, therefore, for example, we might decide to create an entity when, instead, we should decide to update it. The decision made is wrong because the “create” message has not been applied yet in MDM due to JMS asynchronous nature (the create message is waiting in a JMS queue to be processed).

Option B allows us to make trusted decisions because the changes in MDM are applied in the context of the transaction. This will require:

* Sequence the messages that require sequencing, to avoid concurrent transactions happening over the same objects in MDM.
* This sequencing requirement is going to be implemented, so the decisions made by 1 transaction will be made over a trustful MDM current state, knowing that there are no “inflight” transactions pending to be commited.

**NOTE: The sequencing key is yet to be defined, and it will depend probably on the use case. Some messages will be sequenced by ICP#, some by Serial#, and others by the combination Serial# and ICP#**

### 7.1.4. About the outbound pattern to collect file pieces from MDM and send the file

It has been decided that MDM will produce consumption report pieces, published in JMS queues. Mulesoft will collect those pieces by reading from the JMS queues, transform the messages, form the final file and send the file to the destination trader.

### 7.1.5. About the usage of 1 single JMS queue out of MDM

It has been decided that MDM is going to publish the messages in 1 single output queue, probably clustered. This queue will be common to all the output formats and all the Traders. Mulesoft will distinguish

* Which trader is that message belonging to
* Which format is that message going to be transformed to (HHR, RR, etc..)
* Which file that message belongs to

By examining the payload of that message.

### 7.1.6. About the technology of the JMS Broker

It has been decided that the queues to use when MDM publishes the messages are going to be provided by Oracle, using the integrated Weblogic JMS Broker in the Weblogic server. In order to have HA and scaling topology, these queues will be probably set in a cluster mode (distributed queues).

### 7.1.7. About the attributes of logging

The attributes used in the logging framework were discussed in a workshop and the final decision is reflected in the list that can be found in the logging section.

### 7.1.8. About the attributes of auditing

The attributes used in the auditing framework were discussed in a workshop and the final decision is reflected in the list that can be found in the auditing section.

### 7.1.9. About the destination of logging

Logging messages will go to files stored in the file systems. There will be as many log files as nodes in the cluster, as the destination directory of the log files is not shared across the nodes.

The logs will rotate and appended with a sequence number when the maximum size is reached.

### 7.1.10. About the destination of auditing

Auditing messages containing auditing events and business meaningful messages are going to be stored in a Database, in order to facilitate the extraction and the usage of them.

### 7.1.11. About the Primary Keys

The entity mapping done between SAP, COUNTIES and Registry with the entities managed in MDM is governed by a pattern of composing the Unique Keys in MDM with values coming from the SAP, REGISTRY and COUNTIES messages, plus some sequence numbers to guarantee uniqueness.

That table can be found in section “Unique Keys and Entity Mapping” section.

### 7.1.12. About the new format of SAP

It has been decided that SAP is going to change the file format produced for this project. The new file format will be named “v4”.

This new file format will:

* Contain information relevant to manage the reversal scenarios.
* Contain information about manufacturer, model and owner codes of the devices.

### 7.1.13. About the lookups in Counties

It has been decided that the Mule logic will process only the Installation file coming from Counties. We won’t need to process the rest of the files from Counties because the information that they contain can be derived from the installation file + lookup tables. This information is related to Manufacturer, Make. We will have a lookup in Mule being the Key the "CategoryCode" and the value a pair [Manufacturer],[Model].

### 7.1.14. About the real time call to the registry using the API instead of the Web service

It was decided that we will use the API realtime call to the registry in order to obtain realtime information from the Registry. There was another option to use a Web Service call provided by a 3rd party, but this option was discarded.

The option chosen (API Call) has the SLA limitation of 250 calls per minute, and after discussing it in a meeting with <COMPANY NAME>, we agreed that it is going to be difficult to reach that limit, therefore this option was taken.

### 7.1.15. About the Mule Management tool

It has been decided that <COMPANY NAME> will use Anypoint Runtime Manager in the Cloud (CloudHub) as the management console. The decision was made against the usage of MMC (Mule Management Console) because its support will be discontinued.

## 7.2. Risks

The following risks have been identified.

* API Registry calling more times than the SLA, we will get an error. The logic will retry a few times and if the error persist, we will mark it as ERROR.
* The sequencing keys to use are still to be decided, and it may vary depending of the type of message. In particular, the values to use when processing reversal use cases have to be discussed. The options are:
  + Order by ICP# (all the messages that are relevant to the same ICP# will be processed in order or appearance in the file).
  + Order by Serial# (all the messages that are relevant to the same Serial# will be processed in order or appearance in the file).
  + Order by the combination ICP# + Serial# (all the messages that are relevant to the same combination of ICP# and Serial# will be processed in order or appearance in the file).
  + Other key common to all the messages in the file, which is the most conservative sequencing strategy, and it will guarantee that all the messages are processed in order, 1 by 1, without parallel processing of them.
  + Any other key, to be decided, that will make sure that some cases (like reversals processing) are processed sequentially.
* Inability of SAP to send REVERSALS: There has been a decision to produce a new SAP format to include reversal information, but still some workshops have to be taken in place to identify the logic to process the reversal information in the new file.

## 7.3. Assumptions

1. Mule ESB will be stateless and will be used as a orchestration layer . If persistence is required ,then JMS and database will be considered .
2. SAP ,Counties ,Register and MDM 2.1 are the Systems of record.
3. Any issues with the data sets such as asset installation systems must be fixed in the source systems (SAP,Counties ,Registry).
4. The call to Registry will be implemented real time .
5. All the data formatting and validation will be done in Mule .
6. The business rules which change frequently will be managed outside Mule in MDM 2.1 or a rules engine .
7. Mule implementation will be done on 3.8.x runtime.
8. Accurate current volumes and projected volumes will be provided for all the mule applications.
9. Network Infrastructure can cater for increased traffic between Mule Nodes within the Cluster
10. Since Mule Nodes will be clustered, this will require a Load Balancer for HTTP(S) and TCP Network Traffic. The Load balancer will be provisioned by <COMPANY NAME>.
11. While MuleServer can failover in-flight process, the process integrity highly depends on the back-end system capability.
12. The solution depends on suitable infrastructure with the required specifications established during the implementation phase being made available to develop, test and deploy the solution.
13. All data in transit should be considered secure by nature, but not confidential. So secure transmission with no embedded data encryption requirements.
14. Data at rest (excluding passwords and system security related information) will have controlled access via ACL’s (or the like) with no additional encryption on top.

## 7.4. Issues

No Issues identified so far.

## 7.5. Dependencies

The following system contracts have to be provided as dependencies to continue the detail design of the mule applications:

|  |  |  |  |
| --- | --- | --- | --- |
| System | Contract | Contract Type | Contract Definition |
| SRM (Disco/Reco) | SRM Web Service Definition | WSDL | TBC.wsdl |
| SAP | Delta changes - Installation File | XML | [http://sharepoint/<COMPANY NAME>/projects/160701ICEPhase2HHR/Documents/Designs/ZDM\_INSTALLED\_ASSET04.XML](http://sharepoint/metrix/projects/160701ICEPhase2HHR/Documents/Designs/ZDM_INSTALLED_ASSET04.XML) |
| Registry | Registry API | Swagger | <https://emi.portal.azure-api.net/docs/services/56a92d5074ff88075485e86d/operations/56fd9f8fea9dce11ec5eee7f>  <https://emi.portal.azure-api.net/docs/services/56a92d5074ff88075485e86d/export?DocumentFormat=Swagger&ApiName=ICP%20connection%20data> |
| Registry | NOTIFICATION FILE | CSV | TBC.xlsx, or TBC.pdf |
| Registry | CS FILE | CSV | TBC.xlsx, or TBC.pdf |
| Registry | AW FILE | CSV | TBC.xlsx, or TBC.pdf |
| Per Trader | BMR Request File | XML | TBC.xml (XML Schema) |
| Per Trader | BMR Response File | XML | TBC.xml (XML Schema) |
| Per Trader | HHR file | TBC | TBC |
| Per Trader | RR file | TBC | TBC |
| Per Trader | LPE file | TBC | TBC |
| Per Trader | SMURF file | TBC | TBC |
| MDMS 2.1 | Sync Service Point | SOAP | TBC.wsdl |
| MDMS 2.1 | Sync Installation Event | SOAP | TBC.wsdl |
| MDMS 2.1 | Sync Device | SOAP | TBC.wsdl |
| MDMS 2.1 | SQL to obtain payloads of ServicePoint, InstallationEvent, Device given "ICP#" and "SerialNumber" | SQL | TBC.sql |
| MDMS 2.1 | SQL to obtain all the payloads (ServicePoint, InstallationEvent, Device) given a particular "ICP#" | SQL | TBC.sql |
| MDMS 2.1 | SQL to obtain all the payloads (ServicePoint, InstallationEvent, Device) given a particular "SerialNumber#" | SQL | TBC.sql |
| MDMS 2.1 | Launch UT : Web service call to trigger a UT request to MDM | SOAP | TBC.wsdl |
| MDMS 2.1 | SQL to obtain the subscription details to be used to identify the TRADER and the FILE TYPE for the outbound channel given a piece of data read from the JMS queue | SQL | TBC.sql |
| MDMS 2.1 | XML Schema of the messages written in the JMS queue with the consumption report | XML | TBC.xml (XML Schema) |
| MDMS 2.1 | WSDL of the Web Service call to insert a new entry in the TODO list in MDMS. | SOAP | TBC.wsdl |
| COUNTIES | Delta changes - Installation File | XML | TBC |
| Biztalk (Disco/Reco) | Web Service Definition | WSDL | TBC.wsdl |

## 7.6. Constraints

No constraints identified so far.

# 8. Appendix - API Led Approach

## 8.1. Introduction

API-led connectivity from MuleSoft enables companies to meet today’s business and technology challenges and provides a flexible architecture that can rapidly adapt to tomorrow’s needs.

API-led connectivity builds on the central tenets of SOA, yet re-imagines its implementation for today’s unique challenges. API-led connectivity is an approach that defines methods for connecting and exposing your assets. The approach shifts the way IT operates and promotes decentralised access to data and capabilities while not compromising on governance. This is a journey that changes the IT operating model and which enables the realization of the ‘composable enterprise’, an enterprise in which its assets and services can be leveraged independent of geographic or technical boundaries.

API-led connectivity calls for a distinct ‘connectivity building block’ that encapsulates three distinct components:

· Interface: Presentation of data in a governed and secured form via an API

· Orchestration: Application of logic to that data, such as transformation and enrichment

· Connectivity: Access to source data, whether from physical systems, or from external services

### 8.1.1. “Three-layered” API-led connectivity architecture

Large enterprises have complex, interwoven connectivity needs that require multiple API-led connectivity building blocks. In this context, putting in a framework for ordering and structuring these building blocks is crucial. Agility and flexibility can only come from a multi-tier architecture containing three distinct layers:

**System Layer:** Underlying all IT architectures are core systems of record (e.g. one’s ERP, key customer and billing systems, proprietary databases etc). Often these systems are not easily accessible due to connectivity concerns and APIs provide a means of hiding that complexity from the user. 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 rationalization of those systems. These APIs will also change more infrequently and will be governed by Central IT given the importance of the underlying systems.

**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. For example, in a purchase order process, there is some logic that is common across products, geographies and retail channels that can and should be distilled into a single service that can then be called by product-, geography- or channel-specific parent services. These APIs perform specific functions and provide access to non-central data and may be built by either Central IT or Line of Business IT.

**Experience Layer:** Data is now consumed across a broad set of channels, each of which wants access to the same data but in a variety of different forms. For example, a retail branch POS system, e- commerce site and mobile shopping application may all want to access the same customer information fields, but each will require that information in very different formats. Experience APIs are the means by which data can be reconfigured so that it is most easily consumed by its intended audience, all from a common data source, rather than setting up separate point-to-point integrations for each channel.

### 8.1.2. Benefits of API-led connectivity

The benefits of thinking about connectivity in this way include:

#### 8.1.2.1. Business Benefits

• IT as a platform for the business: By exposing data assets as services to a broader audience, IT can start to become a platform that allows lines of business to self-serve.

• Increase developer productivity through re-use: Realizing an API-led connectivity approach is consistent with a service oriented approach whereby logic is distilled to its constituent parts and re-used across different applications. This prevents duplication of effort and allows developers to build on each other’s efforts.

• More predictable change: By ensuring a modularization of integration logic, and by ensuring a logical separation between modules, IT leaders are able to better estimate and ensure delivery against changes to code. This architecture negates the nightmare scenario of a small database field change having significant downstream impact, and requiring extensive regression testing.

#### 8.1.2.2. Technical Benefits

• Distributed and tailored approach: An API-led connectivity approach recognises that there is not a one-size-fits-all architecture. This allows connectivity to be addressed in small pieces and for that capability to be exposed through the API or Microservice.

• Greater agility through loose coupling of systems: Within an organization’s IT architecture, there are different levels of governance that are appropriate. The so-called bi-modal integration or two-speed IT approach makes this dichotomy explicit: the need to carefully manage and gate changes to core systems of record (e.g. annual schema changes to core ERP systems) whilst retaining the flexibility to iterate quickly for user facing edge systems such as web and mobile applications where continuous innovation and rapid time to market are critical. Separate API tiers allow a different level of governance and control to exist at each layer, making possible simultaneous loose-tight coupling.

• Deeper operational visibility: Approaching connectivity holistically in this way allows greater operational insight, that goes beyond whether an API or a particular interface is working or not, but provides end-to- end insight from receipt of the initial API request call to fulfillment of that request based on an underlying database query. At each step, fine grained analysis is possible, that can not be easily realised when considering connectivity in a piecemeal fashion.

MuleSoft’s Anypointä Platform is the only solution that allows enterprises to truly deliver on their digital transformation through realizing API-led connectivity. In particular, Anypointä Platform is the only solution that enables end-to-end connectivity across API, service orchestration and application integration needs through a single unified platform. This allows developers to rapidly connect, orchestrate and enable any internal or external endpoint. The result is a 2x to 5x faster time to launch new initiatives, connect systems, and unlock data across the enterprise and a 30% reduction in integration costs.

# 9. Appendix. Environment Configuration Details

The following data will be required to specify the environment design, per environment:

|  |  |
| --- | --- |
| Mule Node 1 | |
| Hostname |  |
| IP Address |  |
| CPUs |  |
| Cores |  |
| RAM |  |
| Swap |  |
| Disc |  |
| Type | Virtual Machine |
| JVM Memory |  |
| HTTP Port used by Mule |  |
| Java Home |  |
| Mule Home |  |
| Mule logs location |  |
| Mule Startup Script |  |
| Mule Agent Port |  |
| Mule Remote ARM Port |  |

|  |  |
| --- | --- |
| Mule Node 2 | |
| Hostname |  |
| IP Address |  |
| CPUs |  |
| Cores |  |
| RAM |  |
| Swap |  |
| Disc |  |
| Type | Virtual Machine |
| JVM Memory |  |
| HTTP Port used by Mule |  |
| Java Home |  |
| Mule Home |  |
| Mule logs location |  |
| Mule Startup Script |  |
| Mule Agent Port | 7777 |
| Mule Remote ARM Port |  |

|  |  |
| --- | --- |
| Active Directory Configuration | |
| Provider URL+port | ldap://sdasdsd.djkje[:389](http://f5-adauth.newcastle.edu.au:389/) |
| UserDn | CN=muleldap\_np,OU=Mulesoft,OU=sysusers,OU=IT Operations,OU=General,OU=Misc,DC=waddf,DC=<company name>,DC=yyy,DC=zzz |
| The password of the console user | asdfasdf |
| usernameAttribute (uid, sAMAccountName can be set) | SAMAccountName |
| The base context within the LDAP tree structure in which the console searches for users. | OU=Others,DC=waddf,DC=<company name>,DC=yyy,DC=zzz |
| A filter expression used to find entries in the LDAP database that match a particular user. | (SAMAccountName={0}) |
| The base context in the LDAP database in which the console will search for users. | OU=Others,DC=waddf,DC=<company name>,DC=yyy,DC=zzz |
| The attribute used to search for users on the LDAP server. | objectclass |
| This is the value of the attribute used to search for users on the LDAP server. | person |
| The DN of the context used to search for groups to which the user belongs. | OU=Mulesoft,OU=Prod,OU=Servers,DC=waddf,DC=<company name>,DC=yyy,DC=zzz |
| A filter expression that finds roles. | (member={0}) |

|  |  |
| --- | --- |
| Mule Database | |
| Hostname |  |
| Port | 1433 |
| Database |  |
| Username |  |
| Password |  |

|  |  |
| --- | --- |
| MDM Database | |
| Hostname |  |
| Port | 1521 |
| Connection String |  |
| Username |  |
| Password |  |

|  |  |
| --- | --- |
| MDM JMS | |
| Hostname |  |
| Port | 7001 |
| Connection String |  |
| Username |  |
| Password |  |

|  |  |
| --- | --- |
| MDM Webservices | |
| Hostname |  |
| Port |  |
| Protocol | HTTPS/HTTP |
| Certificate (in case it is HTTPS) |  |
| Username |  |
| Password |  |

|  |  |
| --- | --- |
| Registry API | |
| Hostname | **emi.azure-api.net** |
| Path | **https://emi.azure-api.net/ICPConnectionData/single/?id={ICP}** |
| Subscription Key |  |

|  |  |  |  |
| --- | --- | --- | --- |
| SFTP Configurations | | | |
| Remote System | Hostname:Port | Path | User:Password |
| SAP |  |  |  |
| COUNTIES |  |  |  |
| Registry |  |  |  |
| Trader BMR Inbound |  |  |  |
| Trader Outbound |  |  |  |

..