<Project Name>

System Architecture Document

Version <1.0>

[Note: The following template is provided for use with the MuleSoft development projects. Text enclosed in square brackets and displayed in blue italics (style=InfoBlue) is included to provide guidance to the author and should be deleted before publishing the document. A paragraph entered following this style will automatically be set to normal (style=Body Text).]

[To customize automatic fields in Microsoft Word (which display a gray background when selected), select File>Properties and replace the Title, Subject and Company fields with the appropriate information for this document. After closing the dialog, automatic fields may be updated throughout the document by selecting Edit>Select All (or Ctrl-A) and pressing F9, or simply click on the field and press F9. This must be done separately for Headers and Footers. Alt-F9 will toggle between displaying the field names and the field contents. See Word help for more information on working with fields.]

Revision History

Date	Version	Description	Author
<dd/mmm/yy>	<x.x>	<details>	<name>

System Integration Architecture Document

1. Introduction

[The introduction of the **System Integration Architecture Document** provides an overview of the entire **System Integration Architecture Document**. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the **System Integration Architecture Document**.]

This document is intended to serve as an initial reference Architecture for future and ongoing developments.

Thus, it should be treated as a living document and tailored over time with<CUSTOMER>'s team needs and experiences from each project.

The architecture is aimed to cover common use cases and provide a quick reference for each approach from an "integration pattern" perspective and describe how developers can leverage MuleSoft platform capabilities to implement those patterns.

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

[This section defines the role or purpose of the **System Integration Architecture Document**, in the overall project documentation, and briefly describes the structure of the document. The specific audiences for the document is identified, with an indication of how they are expected to use the document.]

1.2 Scope

[A brief description of what the System Architecture Document applies to; what is affected or influenced by this document.]

1.3 Definitions, Acronyms, and Abbreviations

[This subsection provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the **System Integration Architecture Document**. This information may be provided by reference to the project's Glossary.]

Term	Definition

1.4 References

[This subsection provides a complete list of all documents referenced elsewhere in the **System Integration Architecture Document**. Identify each document by title, report number (if applicable), date, and publishing organization. Specify the sources from which the references can be obtained. This information may be provided by reference to an appendix or to another document.]

Author	Document Name	Description

1.5 Overview

[This subsection describes what the rest of the **System Integration Architecture Document** contains and explains how the **System Architecture Document** is organized.]

1.6 Assumption

[List of assumptions that where made while writing this **System Integration Architecture Document**.]

2. Architectural Goals and Constraints

[This section describes the software requirements and objectives that have some significant impact on the architecture; for example, safety, security, privacy, use of an off-the-shelf product, portability, distribution, and reuse. It also captures the special constraints that may apply: design and implementation strategy, development tools, team structure, schedule, legacy code, and so on.]

2.1 Context

Expectations:

- <Define Cloud/OnPrem/Hybrid Architecture>
- <Setup environments (sandbox and production)>
- <Setup Identity Management>
- <Review of n use cases>
- <Core Framework definition>
- <C4E>

2.2 Current State

- <CUSTOMER experience developing with Mule.>
- <CUSTOMER subscription (OnPrem | CloudHub | Hybrid, API Management + Analytics?, VPC, etc)>
- <# of expected use cases>
- <Source code tools used, GitHub>
- <Documentation Tools used>
- <CI/CD Tools used>
- <Stakeholders and roles>
- <Mule version and tools to be used>

2.3 Approach and Methodology

The proposed approach is based on the following methodologies/guidelines

- **Domain Driven Design (DDD)** as a design approach to understand the business problem/solution and to develop solid building blocks that compose the application network
    - Domain Events Brainstorming
    - Define Domains, Value Objects, Aggregates, etc
    - Define Bounded Contexts
    - Define Context Mappings
- **API-Led Connectivity** as a reference focused on:
    - Architectural layout, layered approach
    - Taking into account the speed of change

    - Including teams roles separation
- **<C4E** as a final goal to build a solid foundation with the team>

2.3.1 API Led Connectivity (Conceptual Approach)

2.3.1.1 System Layer

System APIs will provide a means of accessing the underlying/core <CUSTOMER> systems (<PeopleSoft, Databases>, etc) exposing that data in a canonical format, while providing downstream isolation 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.

2.3.1.2 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. These APIs perform specific business processes functions and provide access to non-central data and may be built by either Central IT or Line of Business IT.

2.3.1.3 Experience Layer

Data is now consumed across a broad set of channels/teams, each of which want access to the same data but in a variety of different forms. 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.

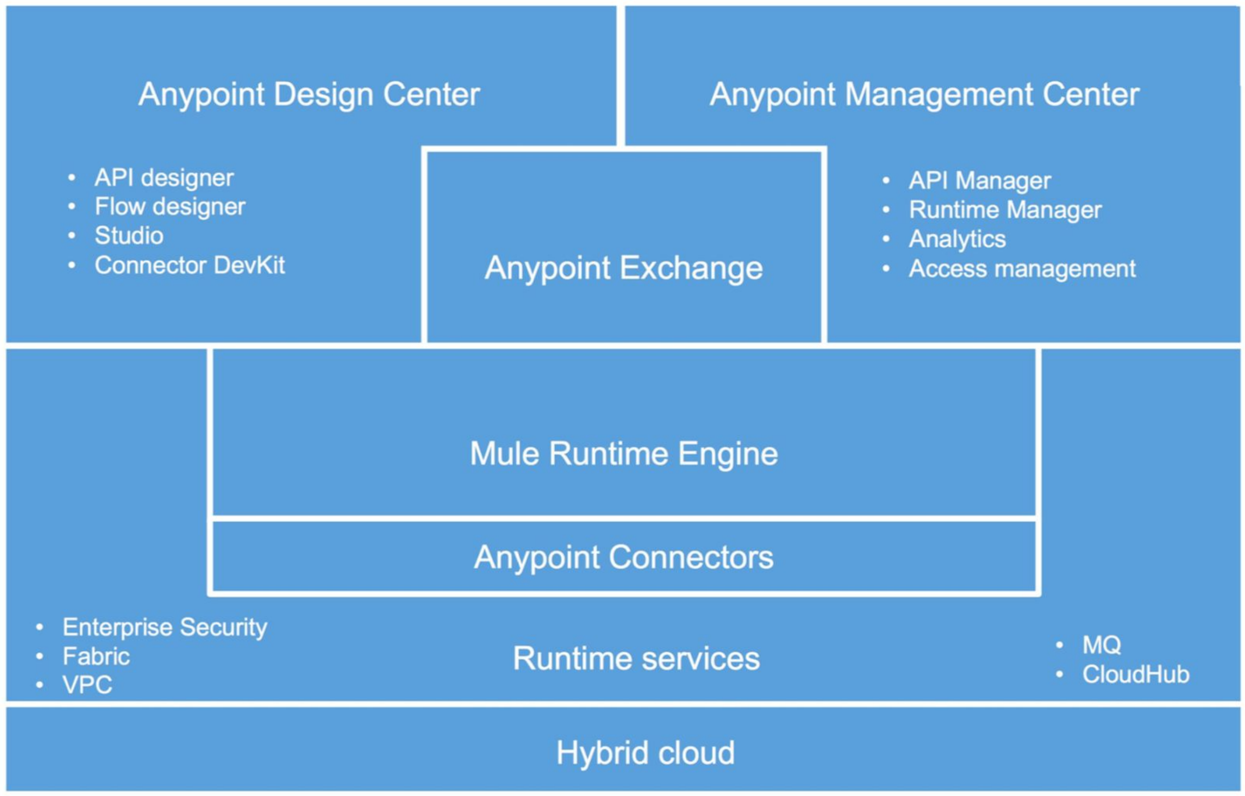
The idea of applying this approach is to have scalable and reusable services, lower maintenance costs, faster time-to-market, flexibility, agility and a low learning curve.

2.3.2 Mapping with <CUSTOMER> Methodologies

SCRUM

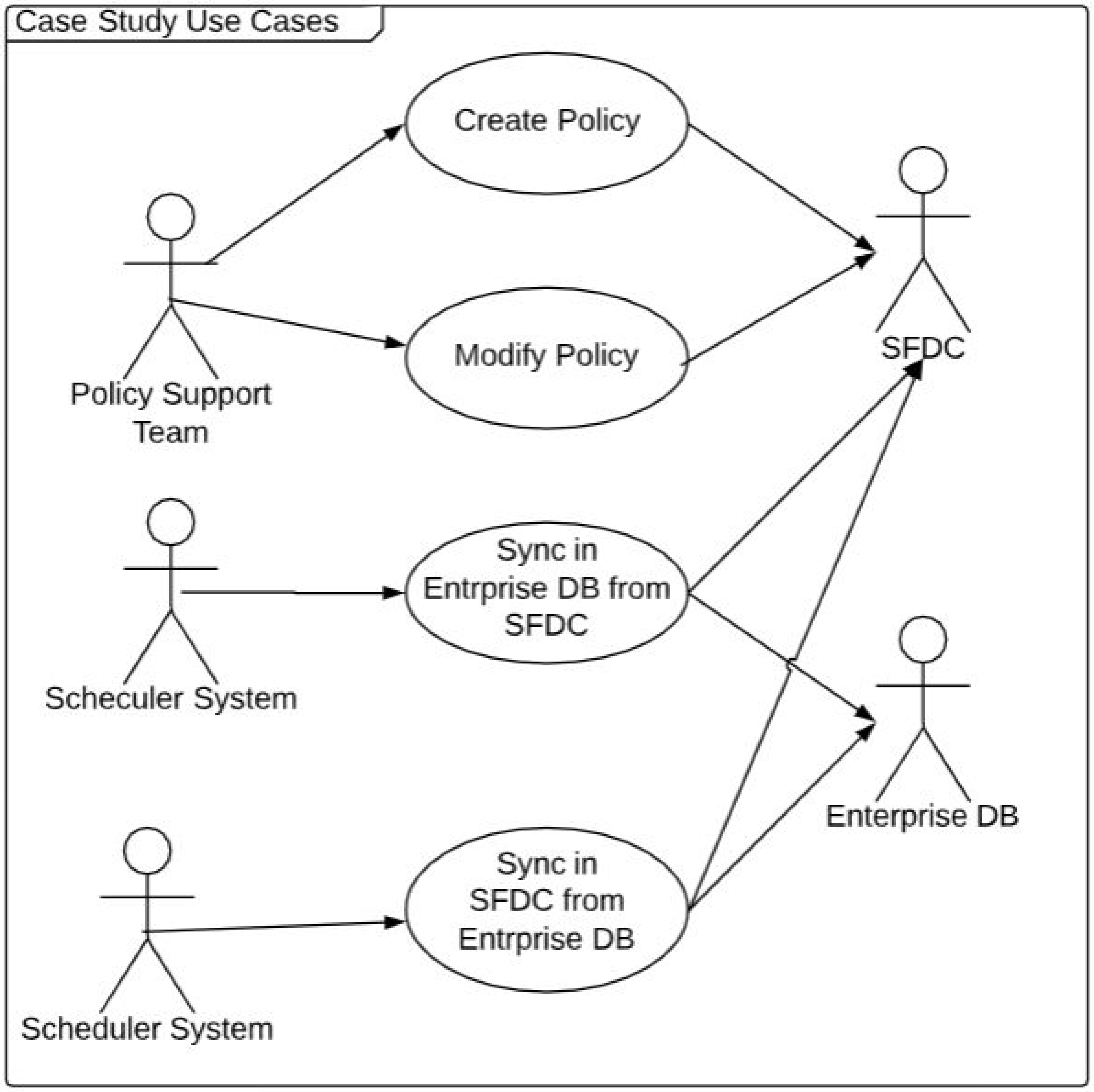
3. Architectural Representation

[This section describes what software architecture is for the current system, and how it is represented. Of the **Use-Case**, **Logical**, **Process**, **Deployment**, and **Implementation Views**, it enumerates the views that are necessary, and for each view, explains what types of model elements it contains.]



4. Use-Case View

[This section lists use cases or scenarios from the use-case model if they represent some significant, central functionality of the final system, or if they have a large architectural coverage—they exercise many architectural elements or if they stress or illustrate a specific, delicate point of the architecture.]



4.1 Use-Case Realizations

[This section illustrates how the software actually works by giving a few selected use-case (or scenario) realizations, and explains how the various design model elements contribute to their functionality.]

The system has Policy management functional requirement. The policy use cases involve interaction between Policy Support Team (actor) and SFDC. The use case realization need to exhibit using logical view to detail out use case.

The system synchronize data between enterprise database and SF and vice versa. The use case realization need to exhibit using logical view to detail out use case.

5. Logical View

[This section describes the architecturally significant parts of the design model, such as sequence diagrams. You should introduce architecturally significant sequence diagrams are typically associated with use case realizations in the Logical View of the system under development.]

5.1 Overview

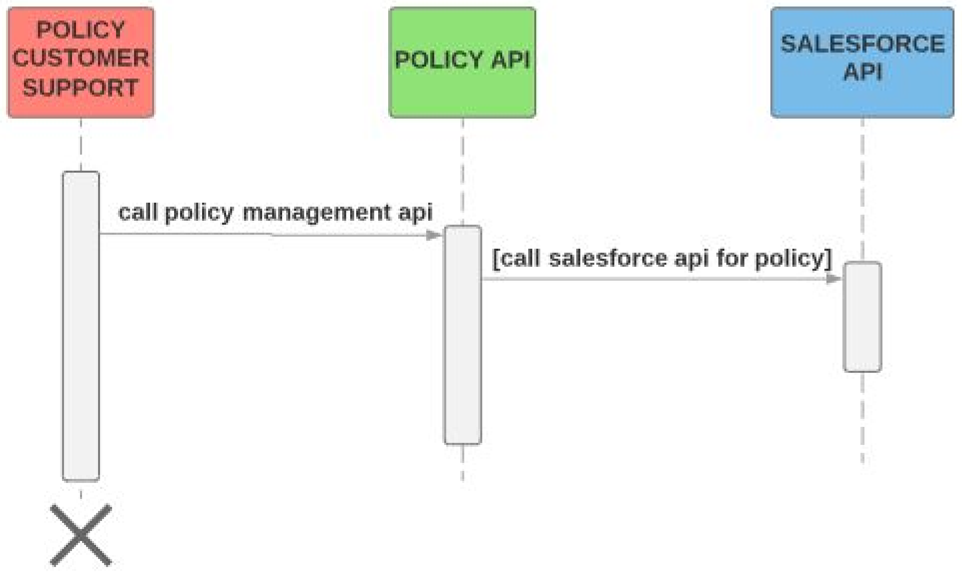
[This subsection describes the overall design model in terms of message exchange between different processes or objects that live simultaneously.]

5.2 Architecturally Significant Logical Flows

[For each significant processes, include a subsection with details about flows with exchange of messages between components.]

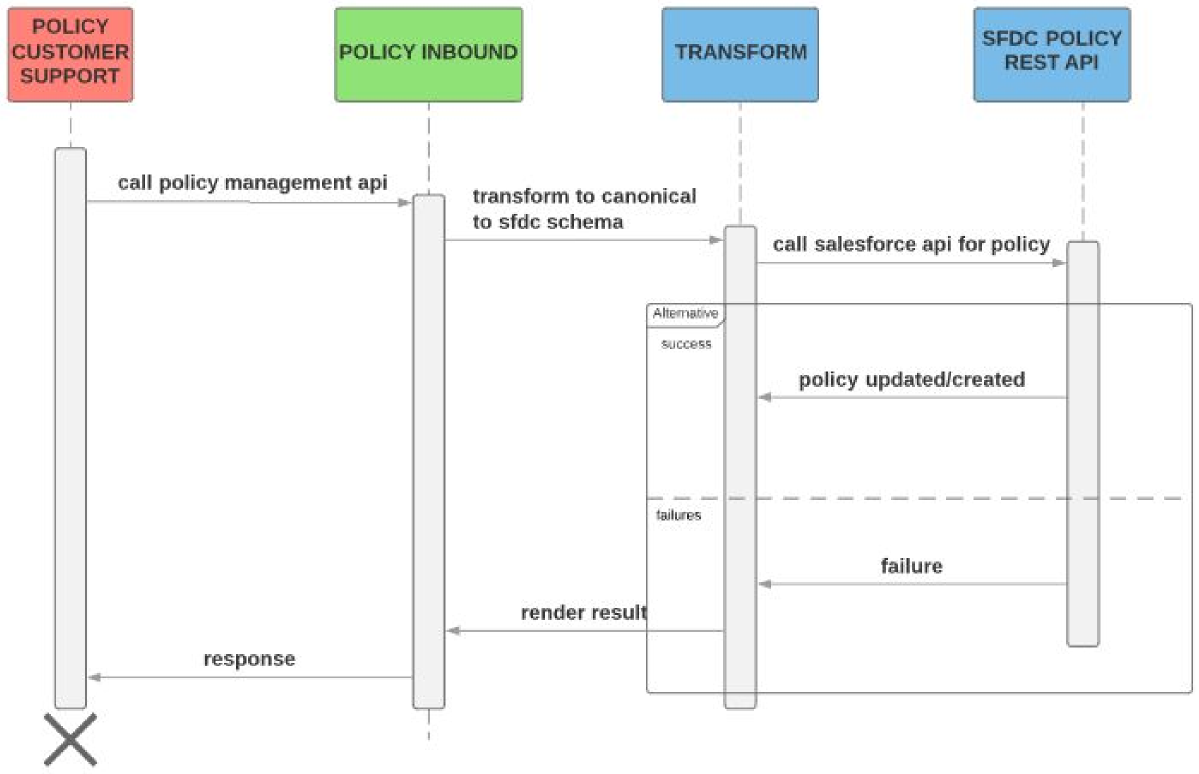
The policy creation and modification is performed policy management API.

Policy management interaction -

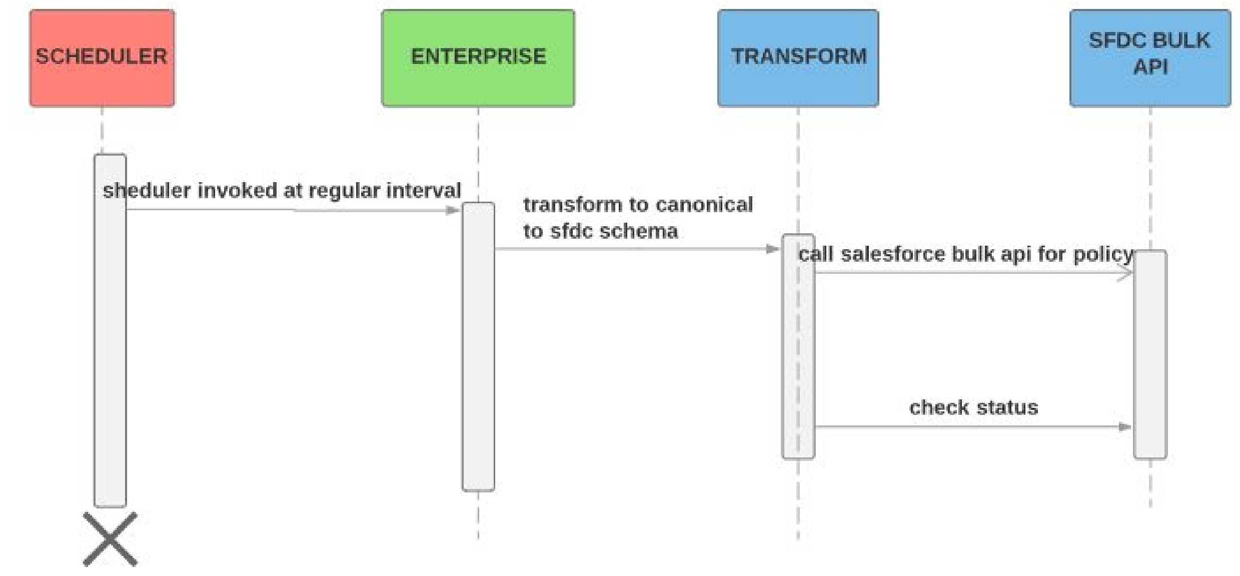


The Policy API is responsible for creating and modifying policy

Policy API Interaction -



Syncing data between SFDC and Enterprise database


SCHEDULER
ENTERPRISE
TRANSFORM
SFDC BULK API
sheduler invoked at regular interval
transform to canonical to sfdc schema
call salesforce bulk api for policy
check status

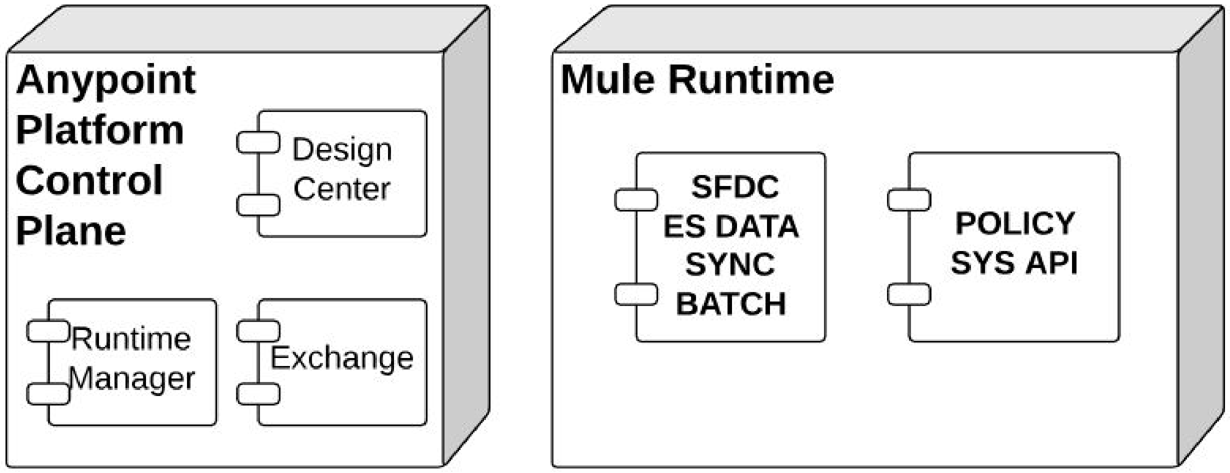
6. Deployment View

[This section describes one or more physical network (hardware) configurations on which the software is deployed and run. It is a view of the Deployment Model. At a minimum for each configuration it should indicate the physical nodes (computers, CPUs, vCores) that execute the software and their interconnections (bus, LAN, point-to-point, VPC and so on.) Also include a mapping of the processes of the **Process View** onto the physical nodes.]

The infrastructure is composed and divided by the following environments:

- Prod
- Sandbox

Example:



7. Process View (optional)

[This section describes the system's decomposition into lightweight processes (single threads of control) and heavyweight processes (groupings of lightweight processes). Organize the section by groups of processes that communicate or interact. Describe the main modes of communication between processes, such as message passing, interrupts, and rendezvous.]

8. Implementation View (optional)

[This section describes the overall structure of the implementation model, the decomposition of the software into layers and subsystems in the implementation model, and any architecturally significant components.]

8.1 Overview

[This subsection names and defines the various layers and their contents, the rules that govern the inclusion to a given layer, and the boundaries between layers. Include a component diagram that shows the relations between layers.]

8.2 Layers

[For each layer, include a subsection with its name, an enumeration of the subsystems located in the layer, and a component diagram.]

9. Data View (optional)

[A description of the persistent data storage perspective of the system. This section is optional if there is little or no persistent data, or the translation between the Design Model and the Data Model is trivial.]

10. Integration Patterns

<Describe the patterns to be tackled>

Examples:

- HTTP Request and Reply

  An event occurs in Policy API, you initiate a process in a Salesforce system, pass the required information to that process, receive a response from the Salesforce system, and then use that response data in policy API.

  A call to the Salesforce system require Policy API to wait for a response before continuing processing. The call to the SFDC system is a synchronous request-reply and API has to process the response as part of the same request as the initial call.

  HTTP synchronous call to Salesforce API is best served by Request and Reply pattern.

  Similarly, call to Policy API is performed by external system used by customer support. These systems are expecting response from policy API in same call. The listener for Policy API must serve the response in synchronous mode.

- Asynchronous

  Salesforce Bulk API invoked but flow doesn’t wait for completion of the process. Instead, the SFDC process receives and acknowledges the request and then hands off control back to flow.

11. Software Development LifeCycle

Source Code:

Staging

- Local Development
- Development environment
- Testing

- Intermediate stages (QA/UAT)
- Production

Testing

- Unit
- Integration
- Regression
- Performance
  - Load
  - Stress
  - Soak
- User Acceptance

11.1 APIs LifeCycle

<links>

11.2 Development principles / guidelines

- Mule Development Recommendations
- Naming Conventions

12. Size and Performance

[A description of the major dimensioning characteristics of the software that impact the architecture, as well as the target performance constraints.]

12. Non Functional Requirements and Service Level Agreements

[A description of how the software architecture contributes to all capabilities (other than functionality) of the system: extensibility, reliability, portability, and so on. If these characteristics have special significance, such as safety, security or privacy implications, they must be clearly delineated. The description of service level agreements for the system and as well as dependent system must be documented.]

The customer data should be reliably transferred between enterprise system and Salesforce, Acme is looking for high availability of 99.99%. The load balancing or clustering solution can be considered for high availability and it has to be cost effective.

The response time for API should not be more than 300 milliseconds and throughput expected for API is 20 TPS.

The communication between integration systems should use TLS. It important to select right persistence for cache and batch jobs. Data Sync between ES and SF should be maintained integrity of systems.

Configuration management should kept all credentials secure

12.1 Failure Points Analysis

Example:

Action	Failure (xx process)			
	SFDC	Mule process	Message Broker	Backend app
SFDC	X	Notifications Replay manual process + 24 hrs restriction	Nothing	Nothing
Mule process	Reconnection Strategy	X	Strategy (retry + backup)	Nothing
Message Broker	Nothing	Nothing	X	Nothing
Backend app	Nothing	Nothing	Nothing	X

12.2 Mechanisms

12.2.1 Specific Mechanisms

12.2.1.1 Testing Strategies

Testing Phase	% Test coverage
Unit Testing	
Integration Testing	

12.2.1.1 Performance Testing Plan

12.2.1.3 Performance Testing Tools

12.2.1.4 Performance Testing load scenarios

- Number of Threads − Simulates the number of users or connections to your server application.
- Ramp-Up Period Defines how long it will take JMeter to get all threads running.
- Loop Count − Defines the number of times to execute the test

12.2.1.5 Dashboard of interest in Iterations

https://jmeter.apache.org/usermanual/generating-dashboard.html

Test results from Performance Testing (Iteration 1)

Test results from Performance Testing (Iteration 2)

Test results from Performance Testing (Iteration 3)

Strategies to meet throughput if any

Strategies to meet low latency if any

Strategies to meet concurrency if any

12.2.1.6 Alerts/Notifications

Component	Condition	QoS	Yes/No
Application	CPU Usage Memory Usage Custom Notification alert Exceeds event traffic threshold Secure data gateway disconnected Secure data gateway connected Worker not responding Deployment success Deployment failure	70% 70%	
Server	CPU Usage Memory Usage Server Load Average Server Thread Count		

12.2.1.7 Logging

Logging level for all application is set through - log4j2.xml in << >>

Logging will be performed <<sync or aync>>

Application logger will use log level <<INFO, DEBUG>>

Application error logger will use log level <<INFO, DEBUG, ERROR>>

Logs retention policy

Max log file size

Max rolled files

External logging ? Strategy ?

12.2.1.8 Error Handling

12.2.1.8.1 Validation Strategy

12.2.1.8.2 Error Handling Strategy

12.2.1.8.3 Flow Level Exception Handling

12.2.1.8.4 Application Level Exception Handling

12.2.1.8.5 Global Exception Handling

12.2.1.8.6 APIKit Exception Handling

12.2.1.8.7 HTTP Status Code

Method	HTTP Status for success	HTTP Status for failure
POST	201	
GET	200	

12.2.1.9 Failure recovery strategies

12.2.1.9.1 Reconnection Strategies for connectors

12.2.1.9.2 Retry strategies for flows

12.2.1.9. 3 Redelivery strategies for flows