MuleSoft

Enterprise Architecture

Common Enterprise Integration Patterns

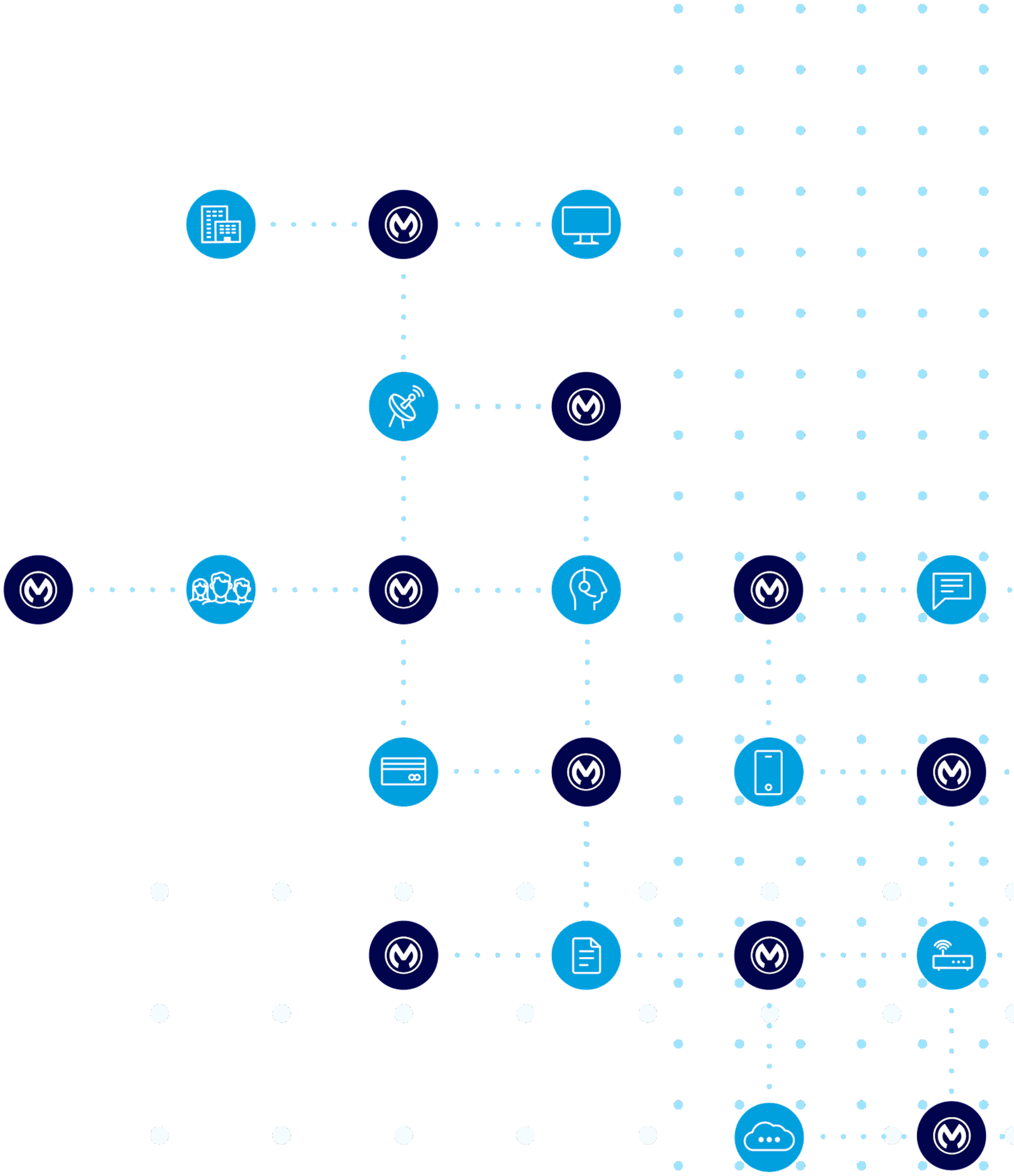


Table of contents

Document Context

Description

Integration could sometimes become a hard and complex problem to solve . To help deal with the complexity of integration design the Enterprise Integration Patterns (EIP) have become the standard way to describe, document,implement and solve complex integration problems.

This document provides guidance through the most common Enterprise Integration Patterns and gives examples of how to implement them irrespective of the integration toolset. This document is targeted for software developers and enterprise architects, but anyone in the integration space can benefit as well.

Purpose

The purpose of a standards and patterns document is to establish a common set of guidelines and best practices for designing and implementing software solutions within an organization. This document typically includes a set of standard practices, coding conventions, and design patterns that are recommended for use across all development projects.

By providing a clear set of standards and patterns, this document helps to ensure consistency, maintainability, and interoperability across all software development efforts. This, in turn, can lead to increased efficiency, reduced costs, and improved quality in software development.

In the context of Enterprise Integration Patterns (EIPs), a standards and patterns document would provide guidance on how to use the various integration patterns to achieve specific integration scenarios. It would outline best practices for implementing EIPs in a consistent and effective manner and would help to ensure that integration solutions are interoperable, scalable, and maintainable over time.

Version History

Revision Number	Revision Date	Summary of Changes	Author

Approvals

This document requires the following approvals. The content of this document is captured after discussions and agreements with the following.

Name	Title, Business Unit/Department
<Enter approver's name>	<Enter approver's designation and department>

Purpose of integration patterns

Enterprise Integration Patterns (EIPs) are important because of following:

1. **Standardization**: EIPs provide a standard set of patterns for integrating applications and systems, which can help ensure consistency and maintainability across the organization.

2. **Scalability**: EIPs can help ensure that integration solutions can scale to handle increasing volumes of data and transactions.

3. **Reusability**: EIPs promote the development of reusable components, which can save time and effort in future integration projects.

4. **Flexibility**: EIPs support a wide range of integration scenarios, such as point-to-point, publish-subscribe, and request-reply, making them highly adaptable to different business needs.

5. **Interoperability**: EIPs promote interoperability between different systems and applications, enabling them to communicate effectively with each other.

6. **Error handling**: EIPs provide standardized patterns for error handling and recovery, which can help reduce the risk of system failures and data loss.

7. **Monitoring and management:** EIPs provide visibility into the flow of data and transactions between systems, which can help with monitoring, management, and troubleshooting.

8. **Cost-effectiveness**: By promoting standardization, reusability, and scalability, EIPs can help reduce the cost and complexity of integration projects.

Overall, EIPs provide a common language and set of best practices for integrating systems and applications, which can help organizations achieve greater efficiency, agility, and innovation in their operations.

Key common integration use-cases

Some of the commonly identified integration use-cases irrespective of any industrial/business context are:

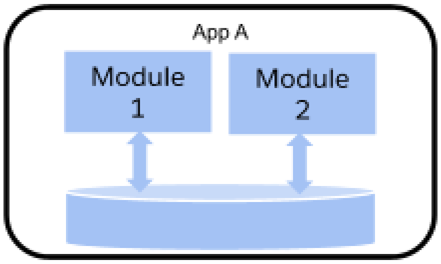
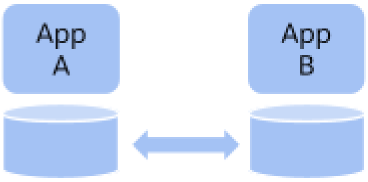
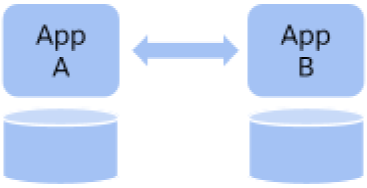
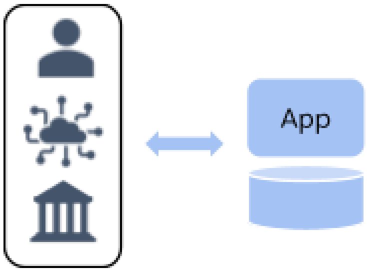
Category	SI. No.	Use-case
Origination	1.	Continuous ingestion of streaming data
	2.	Transform documents and notes to digital data for electronic storage and searching
Movement	1.	Transfer data from System-of-Truth to System-of-Record
Storage	1.	Syncing records to multiple systems
	2.	Moving on-premise data to cloud and vice-versa
	3.	Bulk data migration
Abstraction	1.	Real time data updates / broadcast
	2.	Synchronized, on-demand data transfer / process triggers
	3.	Asynchronous (fire and forget) data transfer / process triggers
Exploration	1.	Data transformation and orchestration during data transfers
	2.	Master data federation across systems
	3.	Aggregating and consolidation data from multiple applications, e.g. Customer 360
	4.	External partner data sharing
Analytics		No integration use cases per se. If information needs to move within or in-between applications, it would be in one of Origination, Storage, Abstraction or Exploration.
Visualisations		

Characteristics of Integration Pattern

Characteristic	Description
Timing/Latency	Does the capability favor batch or real-time operation? For example, more volatile data like deviation, lab equipment readings may require a real-time integration approach if the data is important to the business process
Direction	What kind of data directional support does the capability require – data flowing from source to target, data flowing bi-directional between source and target, data flowing within the firewalls, data flowing outside the enterprise firewalls?
Security	What security mechanisms are supported or required for the capability?
Fault Tolerant	Does the capability need to be fault tolerant? What degree of outage is acceptable?
Scalability	What kind of scalability needs does the capability require?
Reusability	Is the capability reusable across multiple domains/areas/dimensions?
Protocol support	What kind of transport protocol does the capability need to support, if the source and target systems support different mechanisms to send and receive data?
Data format	What kind of data format/structure does the capability need to support, if source systems and target systems support different types of data structure, data models, data definitions
Durability	Does the capability need to provide durable integration?
User Experience	Does the capability have a user experience component in the business process, is a human interaction needed to complete the process?
Skills	What kind of skillsets are required to build, support and enhance the capability?

©MuleSoft, LLC 2020

Integration styles

Style	When to choose?	
In-suite Integration	• Data movement *within* an application/ platform boundary • Data elements *not likely* to be consumed outside of the individual application	
Data Integration	• Synchronize data across application/ platform. • Combine data from multiple application/ platform to provide a unified view	
Process Integration	• Orchestration/chaining of business process interactions across multiple applications • Facilitates completion of an end-to-end business process	
User / Thing / Organization Integrations	• Integration of user interactions, real-life objects, things and organization with the company's business applications • Interactions may be data or process oriented	

©MuleSoft, LLC 2020

Integration Patterns

Overview

Categorically, <Customer> will be using following integration patterns to integrate various source systems and applications:

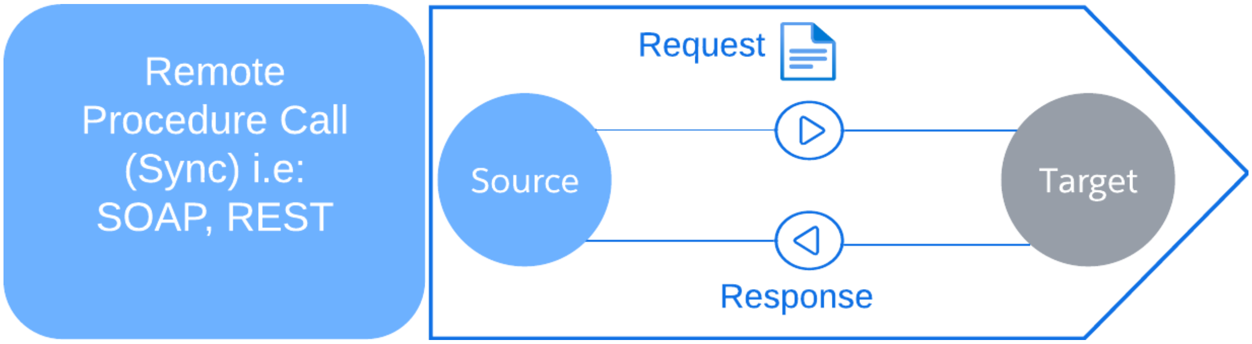
- Real-time integration pattern
- Asynchronous integration pattern
- Batch integration pattern
- Broadcast integration pattern

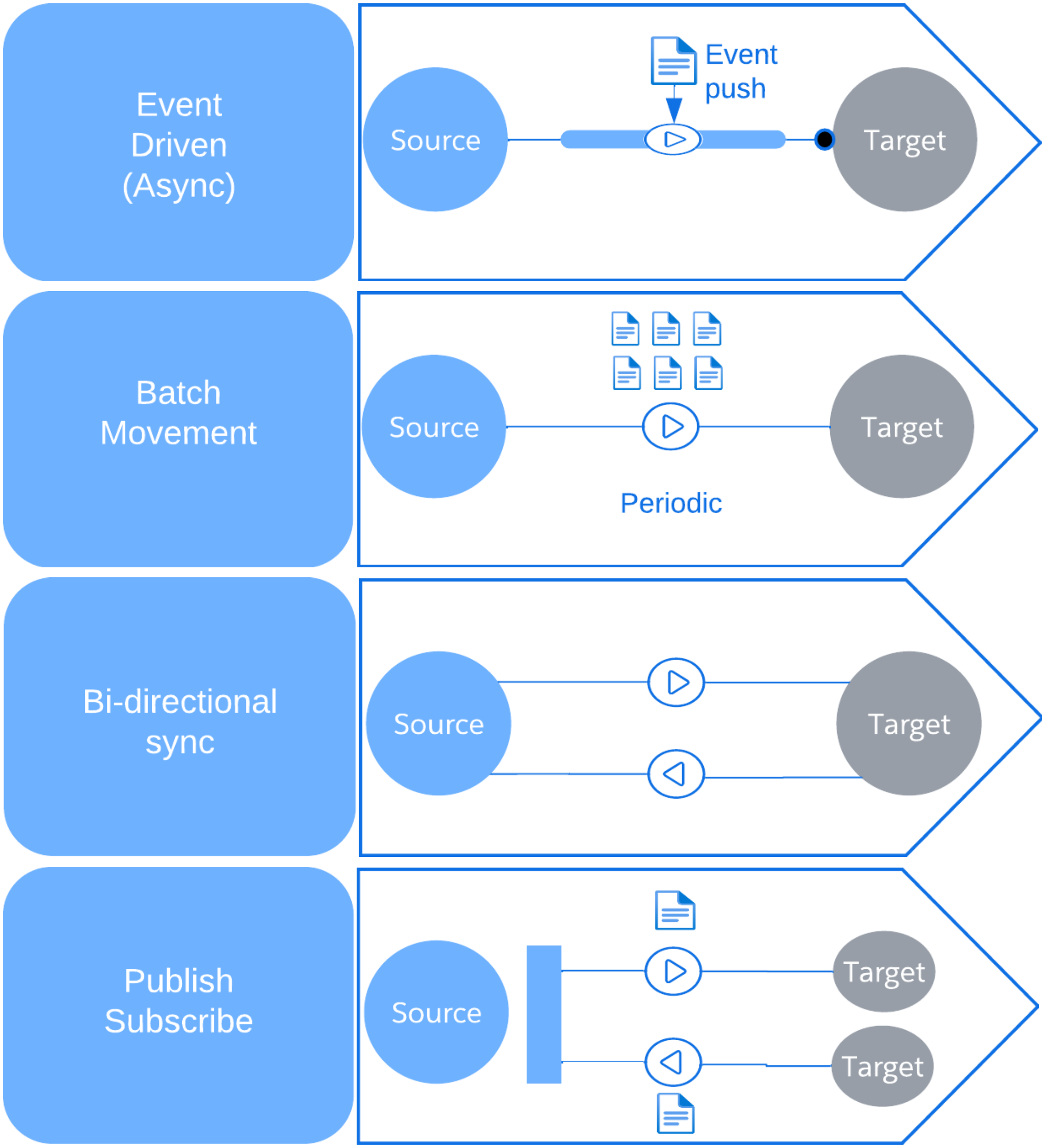
These high-level pattern categories can then further elaborate into multiple foundational and composite patterns.

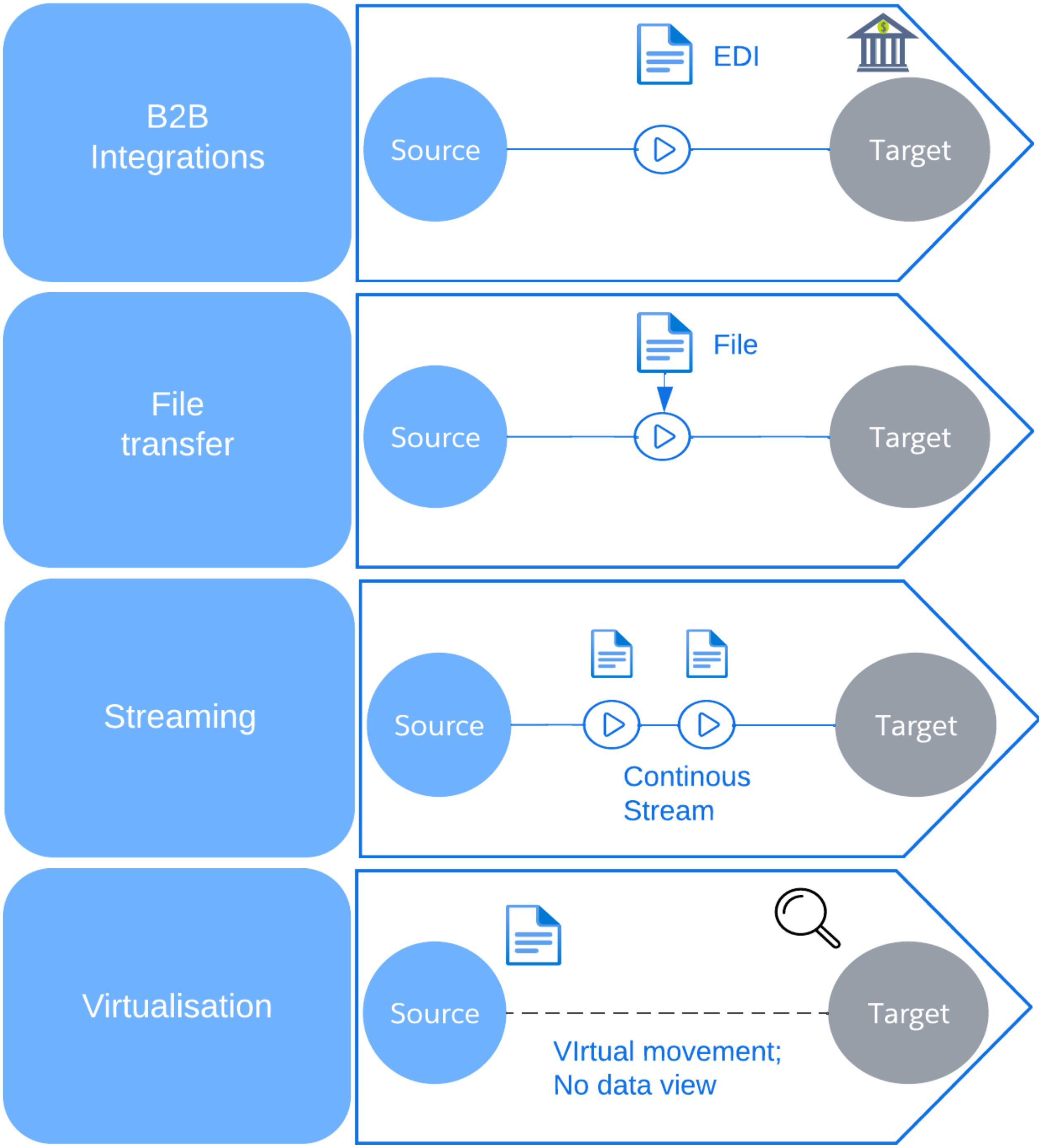
Foundational Patterns

The foundational integration patterns are used as building blocks to construct more complex integration solutions. They provide a standardized approach to common integration challenges and help <Customer> to reduce the complexity and cost of integration, while increasing the agility and flexibility of the platform.

By using these foundational integration patterns, <Customer> can achieve more consistent and reliable integrations, reduce the need for custom development, and accelerate time-to-market for new business initiatives.




Event
Driven
(Async)
Event
push
Source
Target
Batch
Movement
Source
Target
Periodic
Bi-directional
sync
Source
Target
Publish
Subscribe
Source
Target
Target


B2B
Integrations
Source
EDI
Target
File
transfer
Source
File
Target
Streaming
Source
Target
Continous
Stream
Virtualisation
Source
Target
VIrtual movement;
No data view

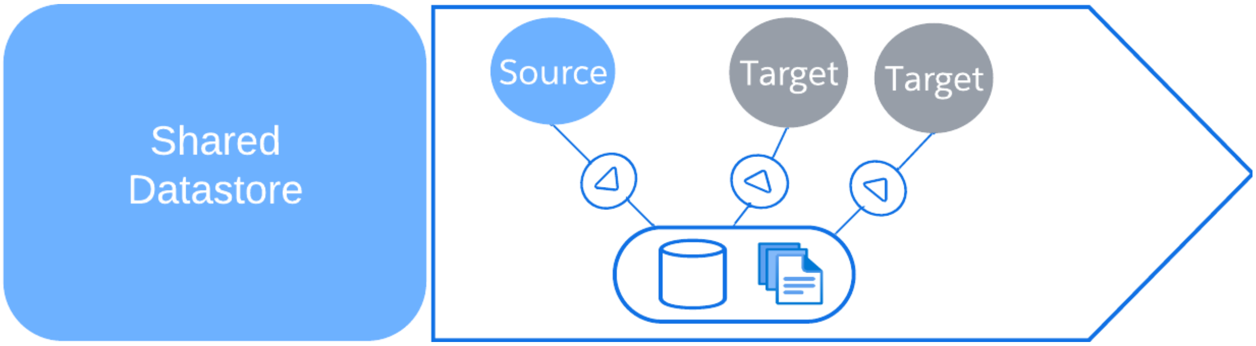


Fig: Common foundational patterns

Composite Patterns

Often business problems may require multiple patterns to be utilized to deliver overall Integration solution.

Combine and/or chain more than one pattern to meet complex business demands.

For example, delivery of a file based on an event that has occurred.

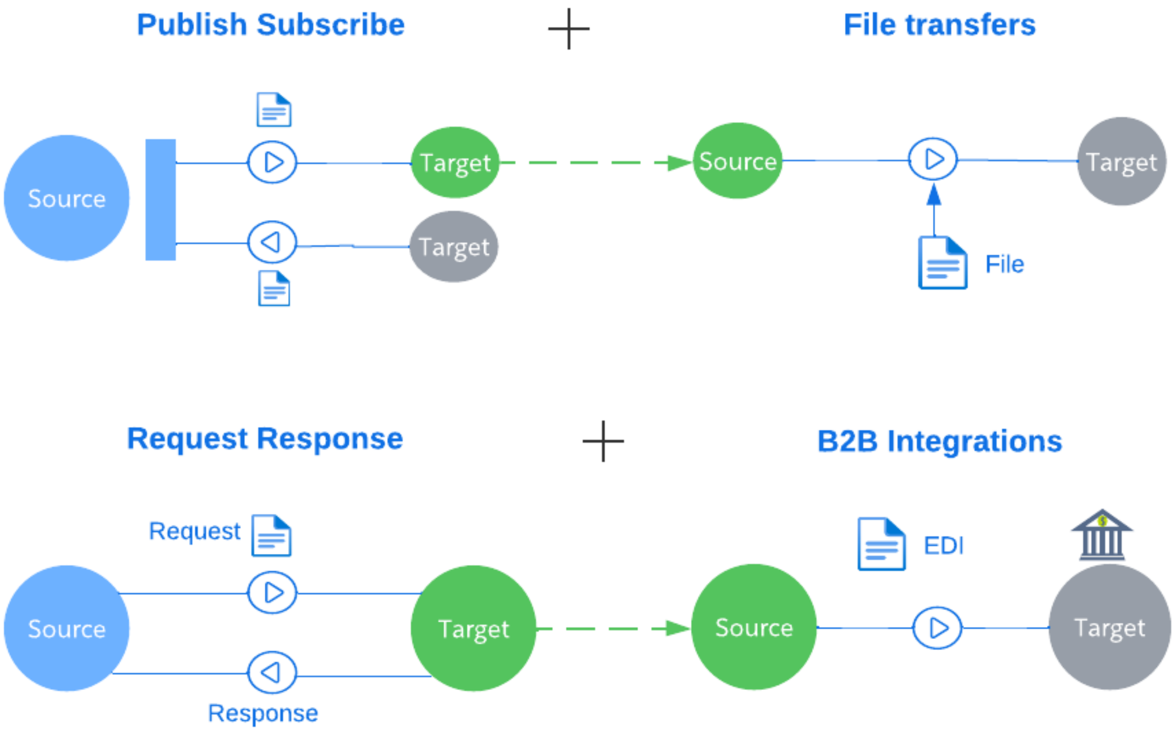


Fig: Composite patterns examples

Decision Trees

Integration patterns and key integration characteristics mapping

✔ full alignment ✔ partial alignment ✖ not aligned

Key characteristic ➜ Pattern ⬇	API-Led	Data driven	Self-service	Discoverable and Consumable	Governable	Compliant
Request Response (Sync)	✔ The Request-Response Sync pattern aligns well with the API-Led characteristic. It involves exposing synchronous request-response APIs that enable real-time integration and communication between systems	✔ The pattern aligns with the Data Driven characteristic as it facilitates the exchange of data between systems in a synchronous manner, enabling real-time data-driven interactions.	✔ The pattern may not directly align with the Self-Service characteristic, as it typically requires coordination between system owners to establish and consume APIs. However, self-service capabilities can be implemented around API documentation, testing, and monitoring to	✔ The Request-Response Sync pattern aligns well with the Discoverable and Consumable characteristic. By providing API documentation, standards, and catalogs, it enables easy discovery and consumption of APIs, making integrations more accessible.	✔ The pattern is governable by implementing API governance mechanisms such as access controls, security policies, rate limiting, and monitoring. It ensures proper control, management, and compliance of the synchronous request-response interactions.	✔ The Request-Response Sync pattern can be implemented in a compliant manner by incorporating security measures, encryption, authentication, and authorization mechanisms. Compliance with data privacy regulations and industry-specific standards can be addressed.

Key characteristic → Pattern ↓	API-Led	Data driven	Self-service	Discoverable and Consumable	Governable	Compliant
			facilitate integration.			
Event Driven	✓ The Event-Driven pattern may not directly align with the API-Led characteristic, as it focuses more on event-based communication rather than traditional request-response APIs. However, APIs can still be utilized to expose event-related functionalities or event registration.	✓ The Event-Driven pattern aligns well with the Data Driven characteristic. It facilitates real-time data processing, analysis, and integration by reacting to events and propagating data changes across systems.	✓ The pattern may not inherently align with the Self-Service characteristic, as event-driven integration often requires coordination between system owners to establish event subscriptions and event handling. However, self-service capabilities can be implemented around event registration and management to enhance developer productivity.	✓ The Event-Driven pattern may not directly align with the Discoverable and Consumable characteristic, as event-driven communication is more event-centric rather than traditional API-centric. However, event-driven systems can provide event catalogs, documentation, and schemas to enhance discoverability and consumability of events.	✓ The pattern can be governed by implementing policies and practices for event management, event routing, security, and monitoring. Proper governance ensures the reliable and secure handling of events across the ecosystem.	✓ The Event-Driven pattern can be implemented in a compliant manner by incorporating security measures, access controls, and data protection practices. Compliance with data privacy regulations, event auditing, and secure event transmission can be addressed.
Bi-Directional Sync	✓	✓	✓	✓	✓	✓

Key characteristic ➜	API-Led	Data driven	Self-service	Discoverable and Consumable	Governable	Compliant
Pattern ⬇						
	The bi-directional sync pattern can be implemented using APIs, allowing systems to exchange data and keep it synchronized in real-time. APIs can be designed and exposed to enable bi-directional data synchronization between systems, aligning with the API-led integration approach.	The bi-directional sync pattern focuses on keeping data synchronized between systems, making it well-aligned with the data-driven integration characteristic. Data is at the core of this pattern, and the synchronization is driven by changes in data within the connected systems.	The bi-directional sync pattern may not inherently align with the self-service characteristic. Implementing and managing bi-directional data synchronization often requires coordination, configuration, and monitoring by integration specialists or IT teams. However, self-service capabilities can still be introduced in terms of defining synchronization rules or triggers within a predefined framework.	The bi-directional sync pattern may not directly align with the discoverable and consumable characteristic, as it primarily focuses on data synchronization rather than exposing discoverable and self-descriptive APIs. However, if the pattern is implemented using APIs, providing clear documentation and standardized interfaces can enhance its discoverability and consumability.	The bi-directional sync pattern can be made governable by implementing appropriate governance mechanisms. This includes defining synchronization rules, establishing data validation and transformation processes, enforcing security and access controls, and maintaining documentation. Effective governance ensures that the synchronization process is managed, controlled, and aligned with	The bi-directional sync pattern can support compliance requirements by enforcing data integrity, security, and privacy measures during the synchronization process. Compliance standards and regulations, such as GDPR or HIPAA, can be incorporated into the synchronization rules and procedures to ensure compliance with data protection and privacy regulations.

Key characteristic → Pattern ⬇	API-Led	Data driven	Self-service	Discoverable and Consumable	Governable	Compliant
					organizational policies.	
Batch Movement	✖ The Batch Movement pattern may not be directly aligned with the API-Led characteristic as it primarily focuses on moving and processing data in batches rather than exposing APIs for real-time integration. However, APIs can still be involved in the pattern for triggering and managing batch processes.	✔ The Batch Movement pattern aligns well with the Data Driven characteristic as it revolves around the efficient movement and processing of large volumes of data. It emphasizes the importance of data integrity, quality, and consistency during batch operations.	✔ The Batch Movement pattern can be aligned with the Self-Service characteristic by providing self-service capabilities for defining and configuring batch jobs. Self-service tools or interfaces can allow users to create, schedule, and monitor batch processes without extensive involvement from IT teams.	✔ The Batch Movement pattern may not inherently align with the Discoverable and Consumable characteristic as it often involves batch files or data transfers rather than exposing discoverable APIs. However, appropriate documentation and standardized file formats can enhance its discoverability and consumability.	✔ The Batch Movement pattern can be made governable by implementing governance mechanisms for managing batch processes. This includes defining policies and procedures for batch operations, ensuring data security and privacy, and establishing monitoring and auditing capabilities.	✔ The Batch Movement pattern can support compliance requirements by incorporating data security and privacy measures during batch operations. Compliance standards and regulations can be addressed through encryption, access controls, data masking, and other relevant practices.
Publish Subscribe	✔ The Publish-Subscribe pattern may not be directly aligned with the API-Led	✔ The Publish-Subscribe pattern aligns well with the Data Driven	✔ The Publish-Subscribe pattern can be aligned with the Self-Service	✔ The Publish-Subscribe pattern aligns well with the Discoverable and	✔ The Publish-Subscribe pattern can be made governable by implementing	✔ The Publish-Subscribe pattern can support compliance

Key characteristic ➜ Pattern ⬇	API-Led	Data driven	Self-service	Discoverable and Consumable	Governable	Compliant
	characteristic, as it focuses on asynchronous messaging and event-driven communication rather than request-response APIs. However, APIs can be used to expose endpoints for subscribing and publishing events.	characteristic. It facilitates the exchange and dissemination of data through the publish-subscribe mechanism, where publishers produce events or messages containing data, and subscribers consume those events for processing or reacting to changes in data.	characteristic by providing self-service capabilities for subscribing and managing event subscriptions. Users can subscribe to specific events of interest, define their own event handlers or workflows, and respond to events without relying heavily on IT teams.	Consumable characteristic. Events can be cataloged or documented, including their schemas, metadata, and associated documentation. This allows subscribers to discover and understand the available events and their usage, making event consumption more discoverable and consumable.	governance mechanisms around event publication, subscription, and management. This includes defining policies and standards for event design, enforcing access controls, monitoring event flows, and ensuring compliance with data governance practices.	requirements by incorporating data security and privacy measures during event processing and communication. Compliance standards and regulations can be addressed through encryption, access controls, and secure event transmission.
B2B Integrations	✔ B2B integrations can align with the API-Led characteristic by exposing APIs to facilitate communication and data exchange	✔ B2B integrations are inherently data-driven, as they involve the exchange of data and information between business partners. The	✔ B2B integrations can align with the Self-Service characteristic by providing self-service capabilities to business partners.	✔ B2B integrations can be made discoverable and consumable by providing clear documentation, standardization of integration	✔ B2B integrations need to be governable to ensure compliance, security, and operational control.	✔ B2B integrations often involve compliance with industry-specific regulations, data protection laws, and security standards. To

Key characteristic → Pattern ↓	API-Led	Data driven	Self-service	Discoverable and Consumable	Governable	Compliant
	between businesses. APIs can be designed and exposed to provide standardized interfaces, enabling seamless integration and interoperability with partner systems.	integration focuses on ensuring the accurate and timely transfer of data in the appropriate format, adhering to specific data requirements and business rules.	This can include self-registration, self-provisioning of integration resources, partner onboarding, and self-management of integration processes. Empowering business partners to handle their integration needs reduces dependency on IT teams and improves agility.	interfaces, and well-defined integration patterns. By offering partner portals, integration marketplaces, or API catalogs, businesses can facilitate the discovery and consumption of B2B integration capabilities.	Governance mechanisms can include partner agreement management, data validation, security measures, and monitoring of integration activities. Establishing governance practices helps maintain control, mitigate risks, and enforce policies within the B2B integration landscape.	ensure compliance, organizations need to implement measures such as data encryption, secure transmission protocols, access controls, and audit trails to protect sensitive information during B2B interactions.
File Transfer	✖ File transfer may not directly align with the API-Led characteristic, as it primarily involves the exchange of files rather than exposing APIs for	✔ File transfer is inherently data-driven, as it involves the exchange and movement of data files between systems or	✔ File transfer can be aligned with the Self-Service characteristic by providing self-service capabilities for initiating and	✔ File transfer may not inherently align with the Discoverable and Consumable characteristic, as it typically involves manual	✔ File transfer can be made governable by implementing appropriate governance mechanisms. This includes defining	✔ File transfer often involves compliance considerations, such as data security, privacy, and regulatory requirements.

Key characteristic ➜ Pattern ⬇	API-Led	Data driven	Self-service	Discoverable and Consumable	Governable	Compliant
	real-time integration. However, file transfer protocols and standards can be used in conjunction with APIs to initiate and manage file transfers.	partners. The pattern focuses on ensuring the secure and reliable transfer of data files, including structured and unstructured data formats.	managing file transfers. This can include self-service portals or interfaces where users can upload, download, and monitor file transfers without extensive IT involvement.	configuration and coordination between sender and receiver. However, by providing clear documentation and standardized file formats, organizations can enhance the discoverability and consumability of file transfer processes.	policies and procedures for file transfer security, access controls, data integrity checks, error handling, and monitoring. Proper governance ensures that file transfers are managed, audited, and compliant with organizational requirements.	Organizations need to implement encryption, secure transmission protocols, access controls, and other relevant measures to ensure compliance with applicable standards and regulations.
Streaming	✓ Streaming may not directly align with the API-Led characteristic, as it primarily focuses on the continuous flow of data rather than request-response APIs. However, APIs can still be	✓ Streaming aligns well with the Data Driven characteristic as it emphasizes the continuous processing and analysis of data in real-time or near real-time. Streaming patterns	✓ Streaming can be aligned with the Self-Service characteristic by providing self-service capabilities for configuring and managing streaming data pipelines.	✓ Streaming may not inherently align with the Discoverable and Consumable characteristic as it often involves real-time data flow rather than exposing discoverable APIs.	✓ Streaming can be made governable by implementing governance mechanisms for managing streaming data pipelines. This includes defining policies for data quality, security,	✓ Streaming implementations need to address compliance requirements related to data privacy, security, and regulatory standards. Organizations should incorporate

©MuleSoft, LLC 2020

Key characteristic → Pattern ↓	API-Led	Data driven	Self-service	Discoverable and Consumable	Governable	Compliant
	used to expose endpoints for streaming data sources, configure streaming pipelines, or provide real-time analytics on streamed data.	enable the ingestion, processing, and delivery of data as it flows, facilitating real-time decision-making and insights.	Self-service tools or interfaces can allow users to define streaming sources, transformations, and destinations without extensive IT involvement.	However, organizations can provide documentation, catalogs, or data catalogs that describe available streaming sources, data formats, and consumption methods to enhance its discoverability and consumability.	access controls, and monitoring the performance of streaming processes to ensure compliance with organizational requirements.	appropriate security measures, data encryption, access controls, and data masking techniques to ensure compliance with applicable regulations.
Virtualization	✓ Virtualization may not directly align with the API-Led characteristic, as it focuses on abstracting and virtualizing data sources rather than exposing APIs for real-time integration. However, APIs can still be utilized to expose virtualized	✓ Virtualization aligns well with the Data Driven characteristic. It involves aggregating, integrating, and presenting data from various sources in a unified and consistent manner. Data virtualization allows real-time	✓ Virtualization can be aligned with the Self-Service characteristic by providing self-service capabilities for accessing and querying virtualized data sources. Users can have self-service tools or interfaces to discover and	✓ Virtualization enhances the discoverability and consumability of data by providing a unified view or abstraction layer over multiple data sources. By offering data catalogs, data dictionaries, or metadata repositories, users can easily discover	✓ Virtualization can be made governable by implementing governance mechanisms for data access, security, and data quality. This includes defining policies for data integration, access controls, data masking,	✓ Virtualization can support compliance requirements by implementing security measures, access controls, and data privacy practices. Compliance standards and regulations can be addressed by incorporating

Key characteristic → Pattern ↓	API-Led	Data driven	Self-service	Discoverable and Consumable	Governable	Compliant
	data services that provide access to consolidated or transformed data.	access to data from multiple systems without the need for physical data replication or migration.	consume virtualized data, reducing dependency on IT teams for data provisioning and access.	and understand the available virtualized data and its usage.	monitoring data usage, and ensuring compliance with data governance practices.	encryption, data masking, and auditing mechanisms to protect sensitive information in virtualized data sources.
Shared Datastore	✓ The Shared Data Store pattern may not directly align with the API-Led characteristic, as it primarily focuses on shared data access rather than exposing APIs for real-time integration. However, APIs can still be utilized to interact with the shared data store, such as querying or updating data through API endpoints.	✓ The Shared Data Store pattern aligns well with the Data Driven characteristic. It enables multiple systems or applications to access and share a common data store, ensuring consistency, integrity, and availability of data across the enterprise.	✓ The Shared Data Store pattern can be aligned with the Self-Service characteristic by providing self-service capabilities for data access and management. Users can have self-service tools or interfaces to query, update, and manage data in the shared data store without relying on IT teams for every data-related task.	✓ The Shared Data Store pattern may not inherently align with the Discoverable and Consumable characteristic, as it often involves direct access to the shared data store rather than exposing discoverable APIs. However, appropriate documentation and data modeling can enhance discoverability and consumability by	✓ The Shared Data Store pattern can be made governable by implementing governance mechanisms for data access, security, and data quality. This includes defining policies and procedures for data governance, access controls, data validation, and monitoring to ensure compliance with regulatory	✓ The Shared Data Store pattern can support compliance requirements by implementing security measures, access controls, and data protection practices. Compliance with privacy regulations, data retention policies, and security standards can be addressed within the shared data store architecture.

©MuleSoft, LLC 2020

Key characteristic →	API-Led	Data driven	Self-service	Discoverable and Consumable	Governable	Compliant
Pattern ⬇						
				providing insights into the structure, semantics, and relationships within the shared data store.	requirements and organizational standards.	

Table: Integration patterns mapped with common key characteristics

Integration patterns and technology mapping

Following table maps the integration patterns with <<Customer>> technology stack.

Technology ➜	MuleSoft	<<Tool>>	<<Tool>>	<<Tool>>
Pattern ⬇				
Request Response (Sync)	✔			
Event Driven	✔			
Bi-Directional Sync	✔			
Batch Movement	✔			
Publish Subscribe	✔			
B2B Integrations	✔			
File Transfer	✔			
Streaming	✔			
Virtualization	✔			
Shared Datastore	✔			

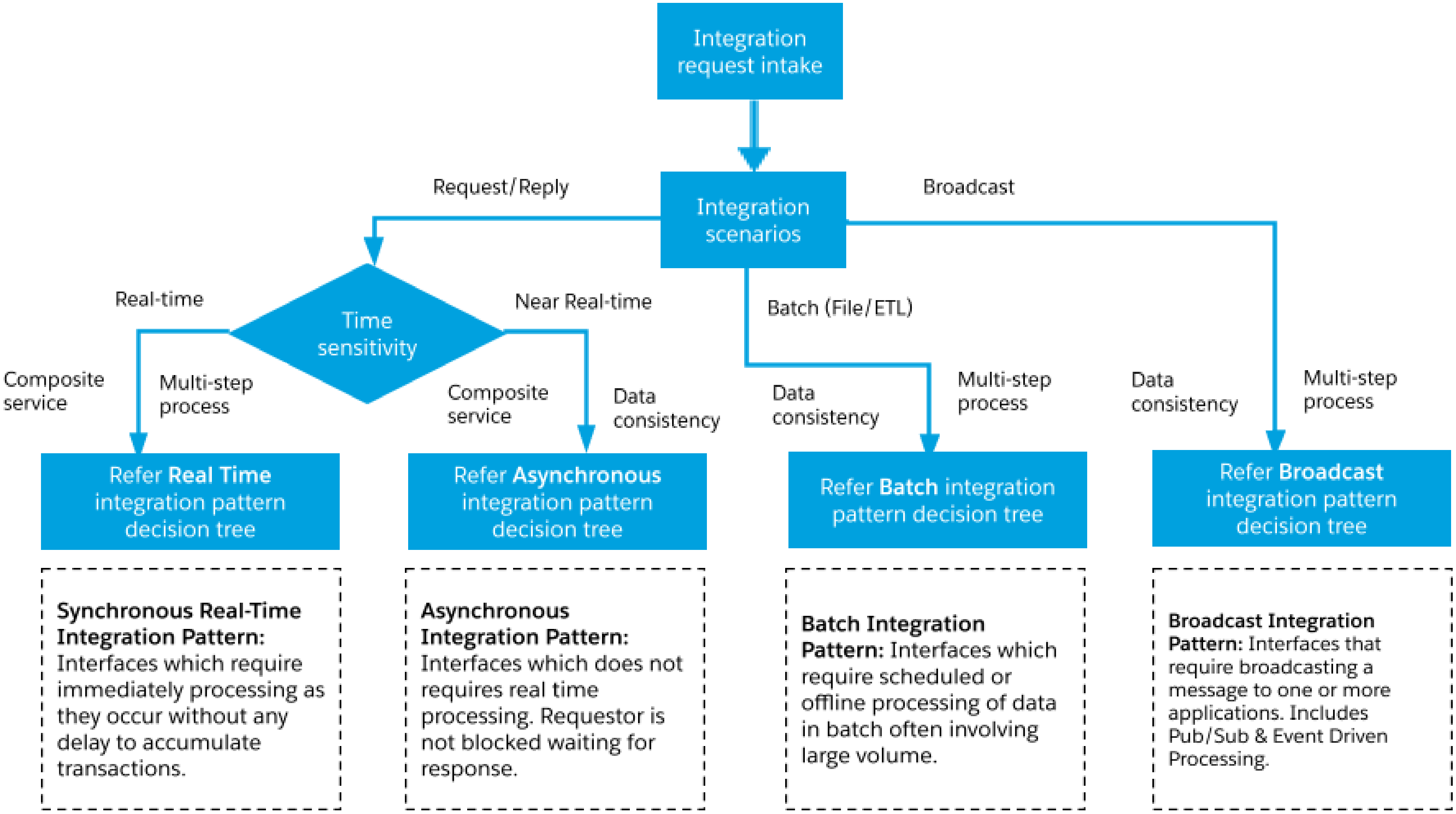
Patterns vs Styles mapping

Following table maps the foundational integration patterns with integration styles.

Technology ➜	In-Suite Integration	Data Integration	Process Integration	User / Thing / Org Integration
Pattern ⬇				
Request Response (Sync)			✔	✔
Event Driven	✔		✔	✔
Bi-Directional Sync		✔	✔	
Batch Movement	✔	✔		
Publish Subscribe			✔	
B2B Integrations			✔	
File Transfer		✔		✔
Streaming		✔		✔
Virtualization	✔	✔		
Shared Datastore	✔	✔		

Integration design patterns decision tree

Following decision tree should be referred to decide the suitability of an integration pattern based on business requirements.



Synchronous Real-Time Integration Pattern: Interfaces which require immediately processing as they occur without any delay to accumulate transactions.

Asynchronous Integration Pattern: Interfaces which does not requires real time processing. Requestor is not blocked waiting for response.

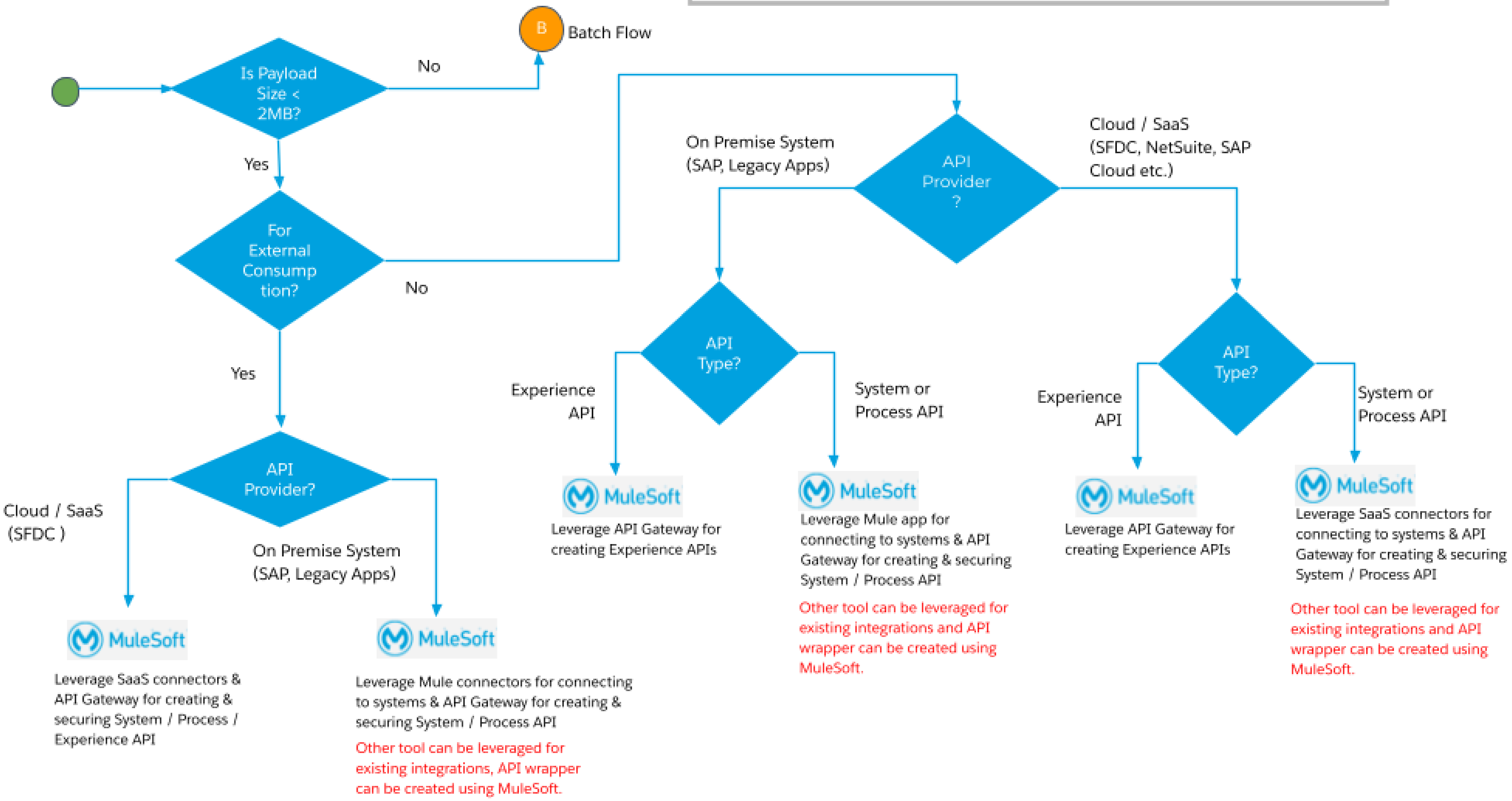
Batch Integration Pattern: Interfaces which require scheduled or offline processing of data in batch often involving large volume.

Broadcast Integration Pattern: Interfaces that require broadcasting a message to one or more applications. Includes Pub/Sub & Event Driven Processing.

Real-time integration pattern flow

Real-Time Criteria

- Composite services and APIs, Real time integrations (Mobile, Web)
- Interface message size in range of <2 MB



B Batch Flow

Is Payload Size < 2MB?

No

Yes

For External Consumption?

No

Yes

API Provider?

Cloud / SaaS (SFDC)

On Premise System (SAP, Legacy Apps)

Leverage SaaS connectors & API Gateway for creating & securing System / Process / Experience API

Leverage Mule connectors for connecting to systems & API Gateway for creating & securing System / Process API

Other tool can be leveraged for existing integrations, API wrapper can be created using MuleSoft.

API Provider ?

On Premise System (SAP, Legacy Apps)

Cloud / SaaS (SFDC, NetSuite, SAP Cloud etc.)

API Type?

Experience API

System or Process API

Leverage API Gateway for creating Experience APIs

Leverage Mule app for connecting to systems & API Gateway for creating & securing System / Process API

Other tool can be leveraged for existing integrations and API wrapper can be created using MuleSoft.

API Type?

Experience API

System or Process API

Leverage API Gateway for creating Experience APIs

Leverage SaaS connectors for connecting to systems & API Gateway for creating & securing System / Process API

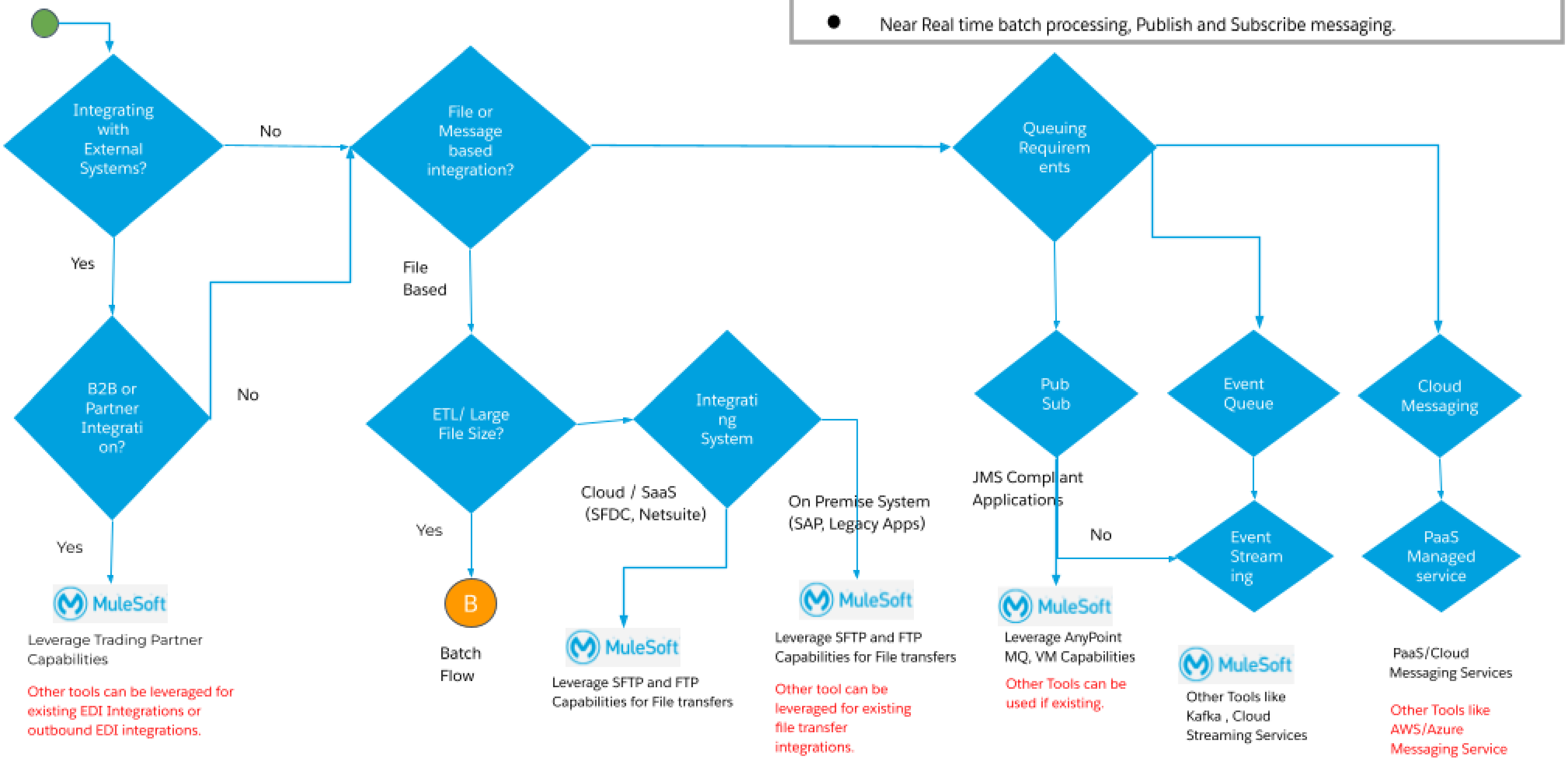
Other tool can be leveraged for existing integrations and API wrapper can be created using MuleSoft.

MuleSoft

Near Real-Time Integration Pattern Flow

Near Real-Time Criteria

- For Interfaces which need persistence & guaranteed delivery
- Interface message size in range of 2 to 10 MB
- Near Real time batch processing, Publish and Subscribe messaging.



Integrating with External Systems?

No

File or Message based integration?

Queuing Requirements

Yes

B2B or Partner Integration?

No

Yes

MuleSoft

Leverage Trading Partner Capabilities

Other tools can be leveraged for existing EDI Integrations or outbound EDI integrations.

File Based

ETL/ Large File Size?

Yes

B

Batch Flow

Integrating System

Cloud / SaaS (SFDC, Netsuite)

MuleSoft

Leverage SFTP and FTP Capabilities for File transfers

On Premise System (SAP, Legacy Apps)

MuleSoft

Leverage SFTP and FTP Capabilities for File transfers

Other tool can be leveraged for existing file transfer integrations.

Pub Sub

JMS Compliant Applications

No

MuleSoft

Leverage AnyPoint MQ, VM Capabilities

Other Tools can be used if existing.

Event Queue

Event Streaming

MuleSoft

Other Tools like Kafka , Cloud Streaming Services

Cloud Messaging

PaaS Managed service

PaaS/Cloud Messaging Services

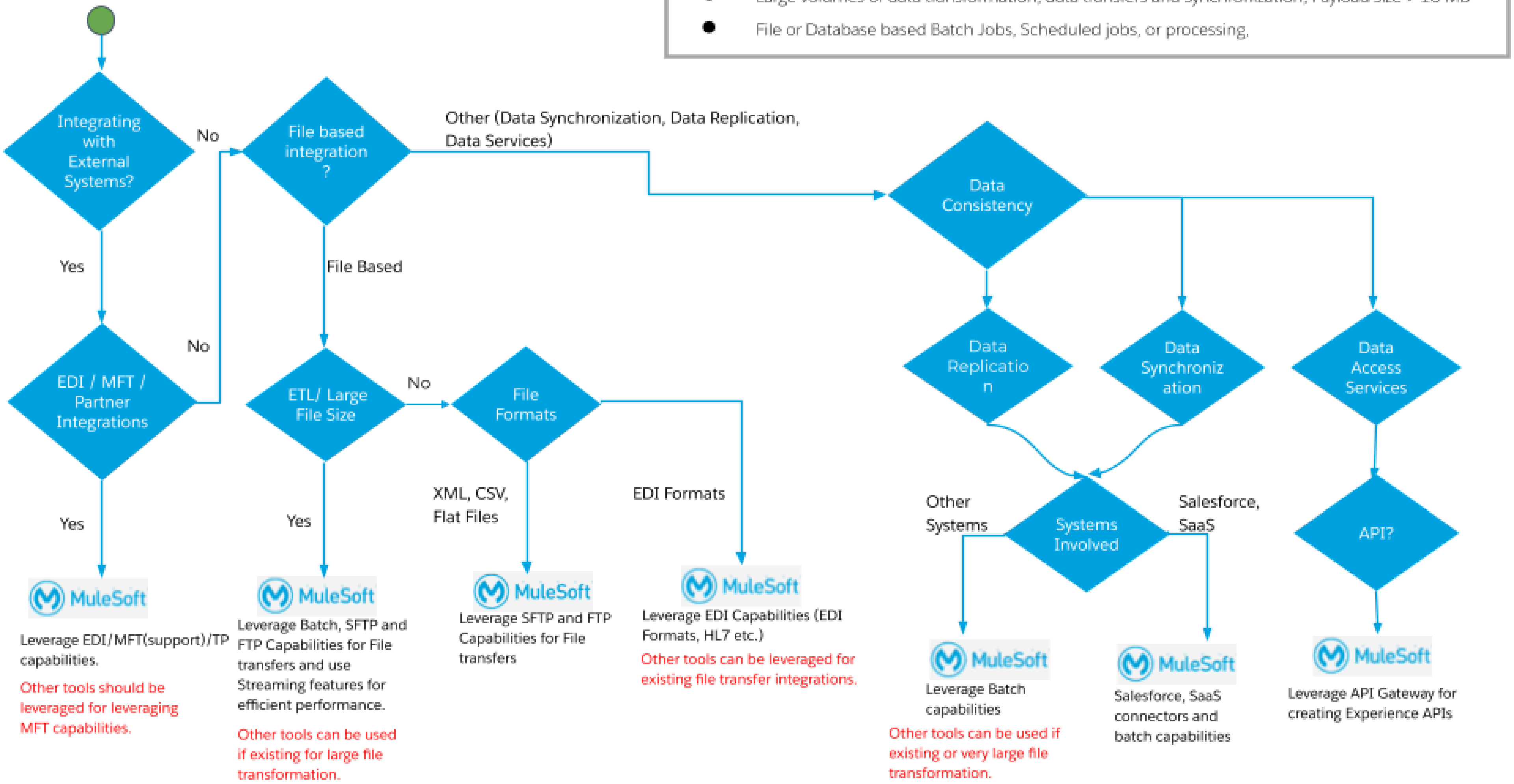
Other Tools like AWS/Azure Messaging Service

©MuleSoft, LLC 2020

Batch Integration Pattern Flow

Batch Criteria

- B2B and EDI Integration, MFT and Partner Integrations
- Large volumes of data transformation, data transfers and synchronization; Payload size > 10 MB
- File or Database based Batch Jobs, Scheduled jobs, or processing.



Integrating with External Systems?

Yes

EDI / MFT / Partner Integrations

Yes

MuleSoft

Leverage EDI/MFT(support)/TP capabilities.

Other tools should be leveraged for leveraging MFT capabilities.

No

No

File based integration ?

File Based

ETL/ Large File Size

Yes

MuleSoft

Leverage Batch, SFTP and FTP Capabilities for File transfers and use Streaming features for efficient performance.

Other tools can be used if existing for large file transformation.

No

File Formats

XML, CSV, Flat Files

MuleSoft

Leverage SFTP and FTP Capabilities for File transfers

EDI Formats

MuleSoft

Leverage EDI Capabilities (EDI Formats, HL7 etc.)

Other tools can be leveraged for existing file transfer integrations.

Other (Data Synchronization, Data Replication, Data Services)

Data Consistency

Data Replication

Data Synchronization

Data Access Services

Systems Involved

Other Systems

MuleSoft

Leverage Batch capabilities

Other tools can be used if existing or very large file transformation.

Salesforce, SaaS

MuleSoft

Salesforce, SaaS connectors and batch capabilities

API?

MuleSoft

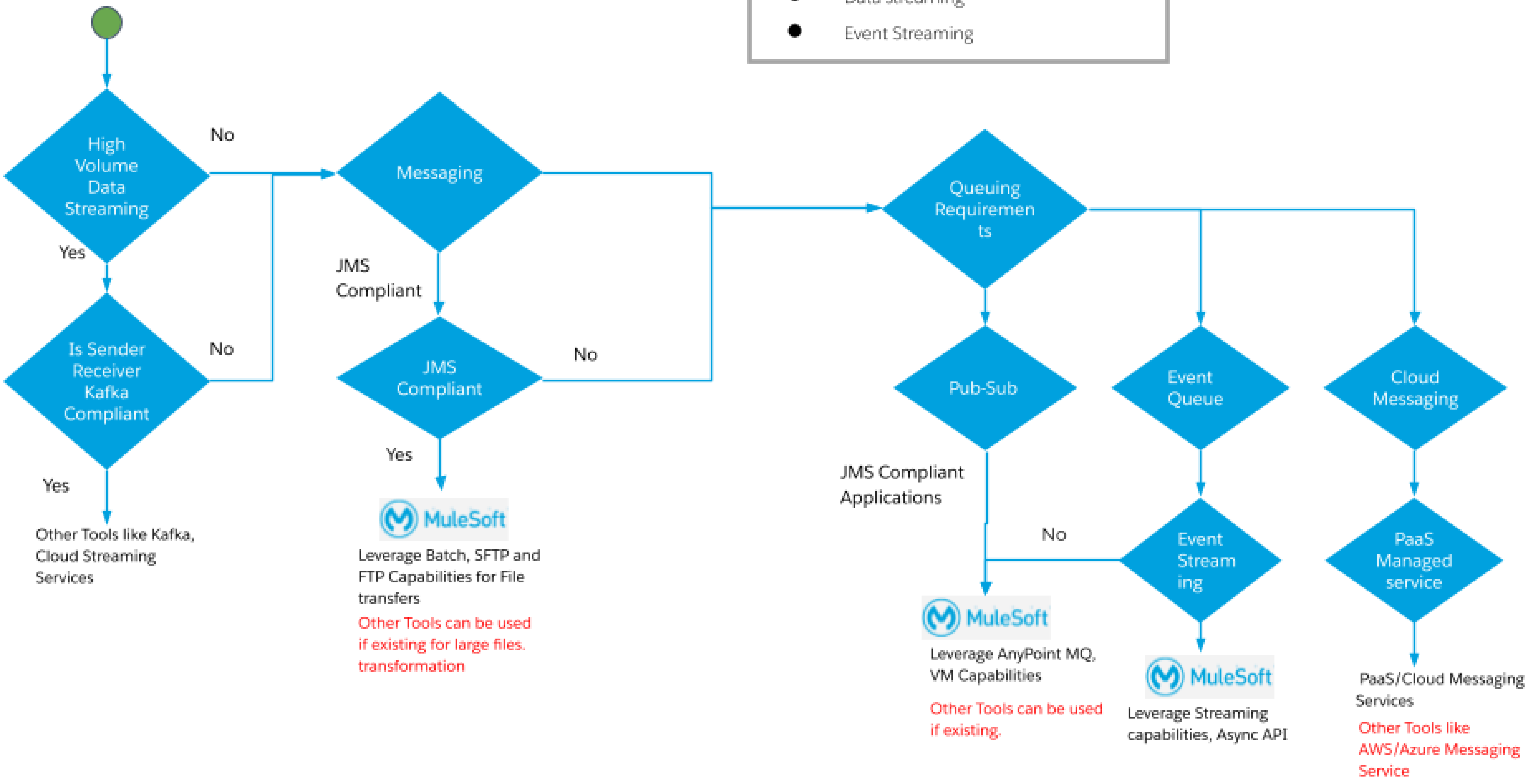
Leverage API Gateway for creating Experience APIs

©MuleSoft, LLC 2020

Broadcast Pattern Flow

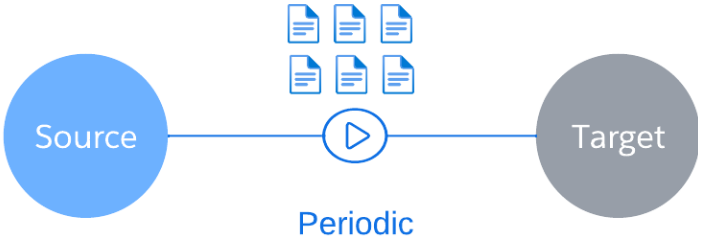
Broadcast Criteria

- Data streaming
- Event Streaming


High Volume Data Streaming
No
Yes
Is Sender Receiver Kafka Compliant
No
Yes
Other Tools like Kafka, Cloud Streaming Services
Messaging
JMS Compliant
JMS Compliant
No
Yes
MuleSoft
Leverage Batch, SFTP and FTP Capabilities for File transfers
Other Tools can be used if existing for large files. transformation
Queuing Requiremen ts
Pub-Sub
Event Queue
Cloud Messaging
JMS Compliant Applications
No
Event Stream ing
PaaS Managed service
MuleSoft
Leverage AnyPoint MQ, VM Capabilities
Other Tools can be used if existing.
MuleSoft
Leverage Streaming capabilities, Async API
PaaS/Cloud Messaging Services
Other Tools like AWS/Azure Messaging Service

Foundational integration patterns

Batch Movement



Use-case scenarios:

The batch movement pattern should be used in following use-case scenarios:

- Data ETL (Extract, Transform, Load)
- Large Volume 'One-Off' Data Migration
- Large Volume 'Periodic' Data Sync
- Data Warehousing, Analytics and Reporting
- Data Archiving
- No Real Time Use Cases

When NOT to use:

Do not use this pattern in following cases:

- Real-Time Scenarios
- Event Oriented Scenarios
- Small Dataset Movement

API-led approach and guidelines:

Batch based data transfer could be achieved using two manners:

- Option 1: MuleSoft Batch Processor a MuleSoft application that utilizes specially designed Batch Processor components.
- Option 2: ETL Tool that already exists as an available option.

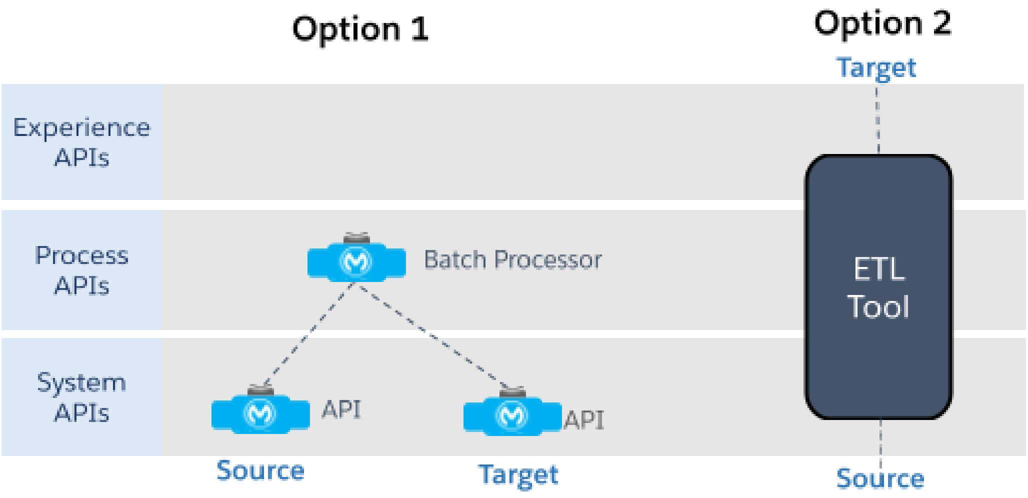


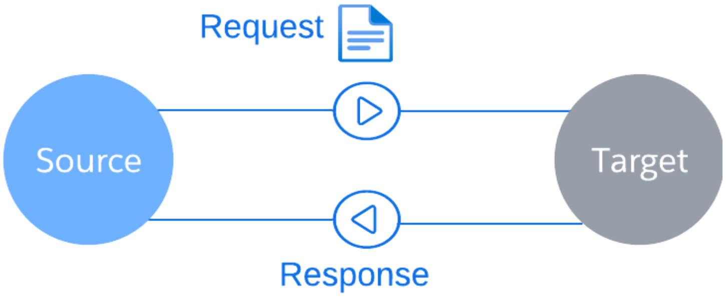
Fig – Batch with API-led options

Considerations during decision making process for tool/pattern selection:

API Led / Reusability Use Case	If the solution use-cases are geared towards, API-Led prefer MuleSoft as platform
Data Volume / Transactions	If large volumes of data need to be moved in P2P manner (2 or more data repositories) then prefer ETL tool. For transactional data, prefer MuleSoft. If MuleSoft Batch Processor is chosen for doing P2P large volume data transfers, then the pattern should be implemented using the MuleSoft Batch development best practices such as Streaming features, dataweave streaming etc.
Real-time vs Scheduled	If real-time, prefer MuleSoft Batch Processor. If scheduled, ETL Tool should be preferred but the solution is doable using MuleSoft schedulers.
Transformation	Prefer ETL tool if complex transformation whereas MuleSoft for message-oriented transformations
Full Load or CDC (change data capture)	If full-load is required in one-off scenarios, prefer ETL tool over MuleSoft Batch Processor
Tool Familiarity	Choose the tool based on familiarity of the tool use-cases / business users / development team that are already familiar with the ETL tool

Performance	ETL tool may perform better in certain scenarios. MuleSoft's Batch processor will require careful design and upscaling allocated 'cores' for performance

Request Response (Sync) | SOAP, REST



Use-case scenarios:

The pattern should be used in following use-case scenarios:

- Sequential Dependency on Response Before Processing Further Steps
- On Demand CRUD Operations / Process Trigger
- Services with quick response time
- Fault Detection and Retries Are Possible

When NOT to use:

Do not use this pattern in following cases:

- Long Running Tasks / Small Timeouts
- No Real Dependency on Request Completion
- Fire and Forget Scenarios

API-led approach and guidelines:

- The Source API could be a System API or a Process API and is called by a higher-level API – either Process API or Experience API.
- Always follow the North-South data transfer direction where an Experience API is in North and a system API is in South. A single system can have both Experience and System APIs.
  Example – From Salesforce to SAP for sending the accounts master data, utilize the Salesforce Experience API with SAP system API. If SAP wants to send the updates to Salesforce then the design should use SAP Experience API.

- It is not mandatory to have Process APIs if there is no business orchestration. Pass-through APIs should be avoided.

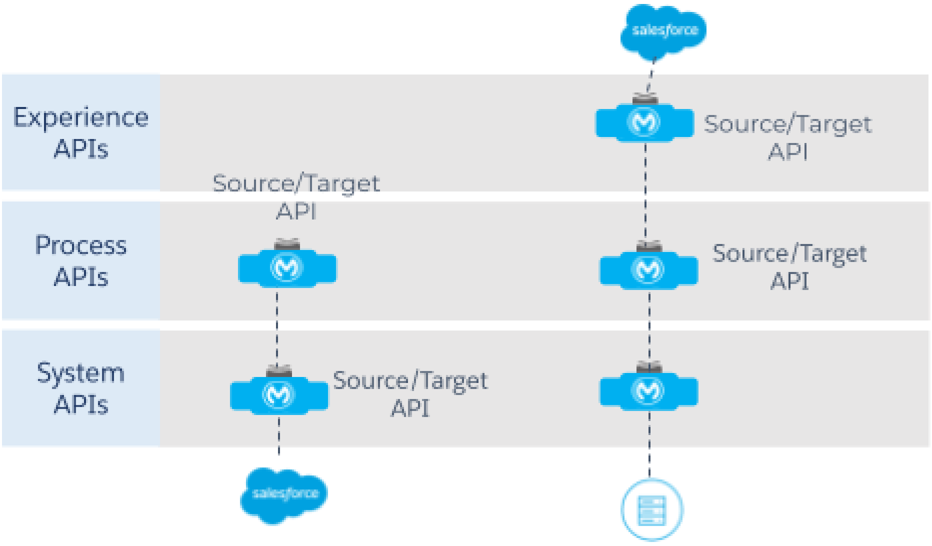
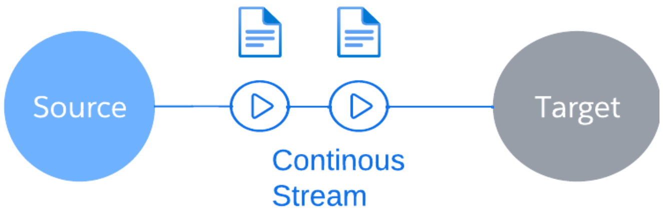


Fig – API-led north-south example for implementing a Sync pattern.

Considerations during decision making process for tool/pattern selection:

Timing/Latency	From the time a request is sent, and the response is obtained, the time lag needs to be carefully determined and the resulting effect on the business process or user experience
Retry / Fault Tolerant	When implementing this pattern, appropriate Retry logic and fault tolerant mechanisms needs to be implemented, when needed
High Reliable / SLAs	The source application needs to be highly reliable where SLAs need to be published
Tighter Coupling	Synchronous calls bring tighter coupling between the source and target applications. Any changes to the source applications directly impacts all target applications
Resource Cost	Making Sync calls are expensive – resources are held for the duration the sync call doesn't complete

File based integration



Use-case scenarios:

The pattern should be used in following use-case scenarios:

- Target system limitations
- Source systems limitations
- Infrastructure limitation
- File Based Data / As-Is File Already Exists
- Minimal Data Transformations

When NOT to use:

Do not use this pattern in following cases:

- Sensitive Data - Unless additional security mechanisms are put in place to protect sensitive data.
- Real Time Scenarios - Because file transfers tend to be inherently time-based or poll-based.
  Exception: Unless triggered by an event driven pattern

  Example: Project cost data form SAP to Clarity

API-led approach and guidelines:

- API-led approach to file transfers, promotes reuse of the process and system APIs for multiple different source systems, partners, and integrations into third party systems.

- A new experience API/application is to be created that accept/send data from/to source/target system in their native format, allowing maximum reuse at the process and system layer.

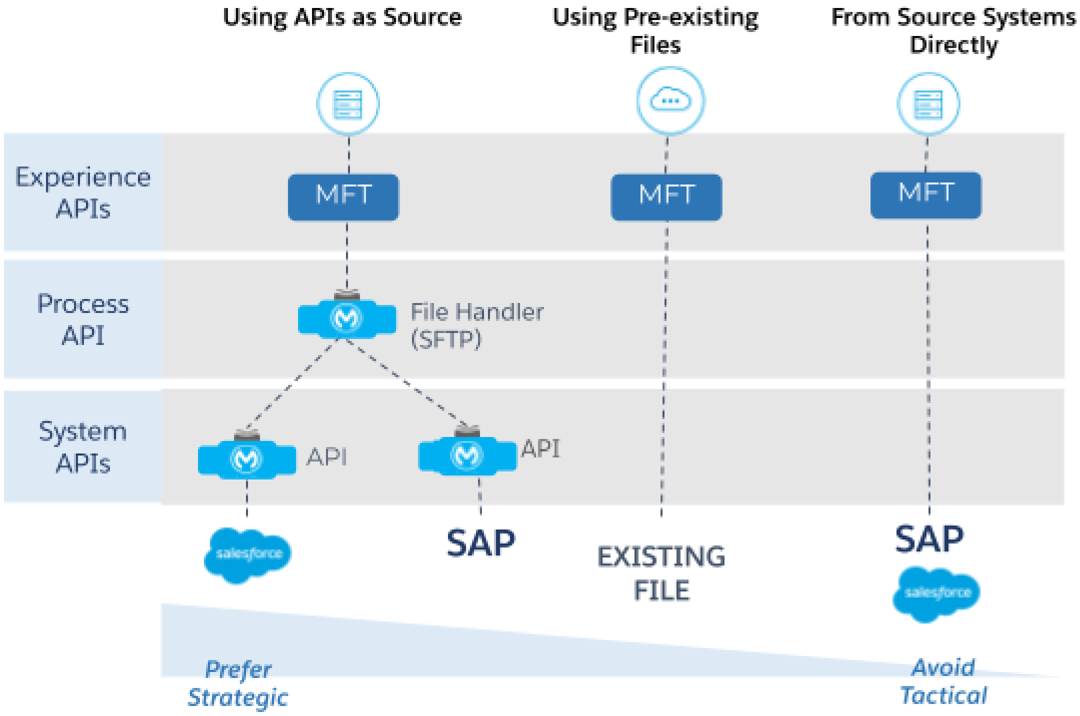


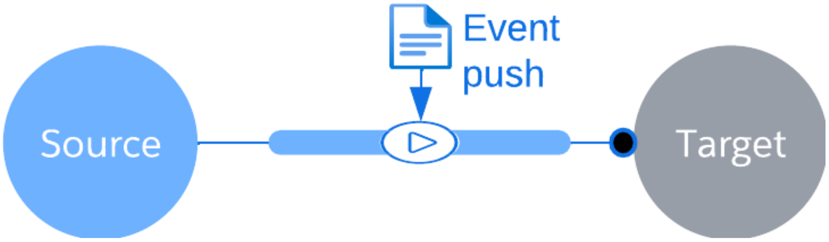
Fig – API-led example for a file pattern

Considerations during decision making process for tool/pattern selection:

Timing / Latency	Polling or timer-based invocations that tend to be non-real time
Data Volume	Limited to the possible file-sizes. Ideal for large volume of data
Computation	Additional computation mechanism required to parse files and process underlying data
Development	Relatively faster development
Maintenance	Significant maintenance effort if connected directly to the source systems
Reusability	Re-usability will be limited to level of file and its content
Durability	High – Data within the files can be persisted and reproduced on demand

Performance	Performance penalties because of IO overheads with native file system operations
Security	File-system storage level security constructs required in addition to securing the in flight data

Event Driven (Async) | MQ



Use-case scenarios:

The pattern should be used in following use-case scenarios:

- Event Based Notification / Status Update.
- Target System Can Wait for the notifications.
- Long Running Tasks in the Source System.
- Near-real Time Data Consumption.
- Reactive and push-based use-cases.
- Back Pressure / Throttling of Events or Requests.
- Durable and Reliable Operations.

When NOT to use:

Do not use this pattern in following cases:

- On Demand operations with sequential dependencies
- Time-sensitive processing

Example; Quality notification to SAP

API-led approach and guidelines:

- Key to utilize an Async pattern is deployment of a messaging system. – such as IBM MQ, Active MQ, AnyPoint MQ.
- A System API is developed in front of the systems that generate/consume data. The data is published to the message system that is handled by an Event Handler

©MuleSoft, LLC 2020

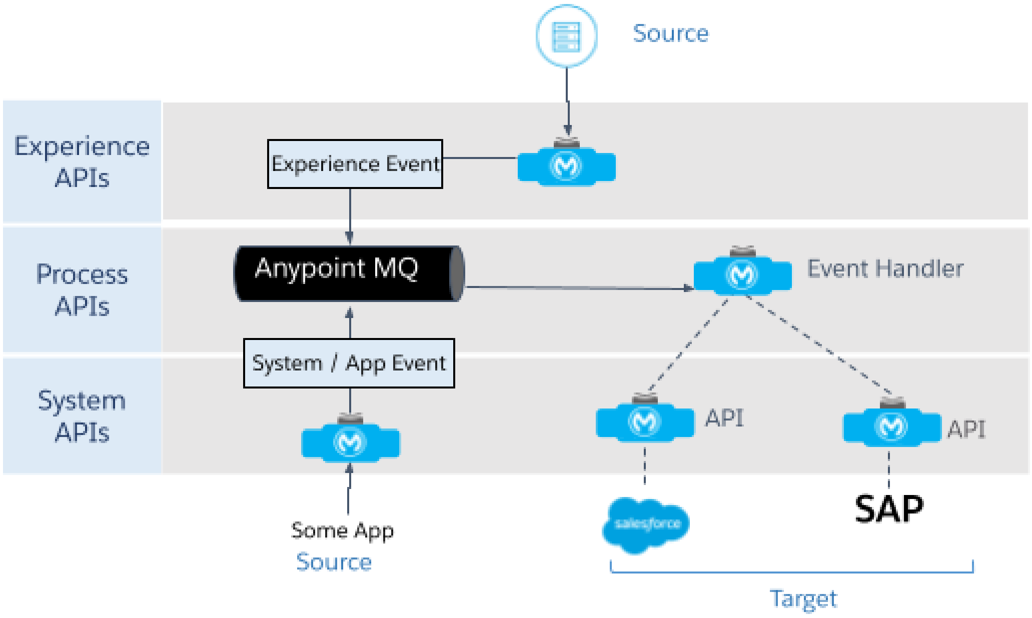
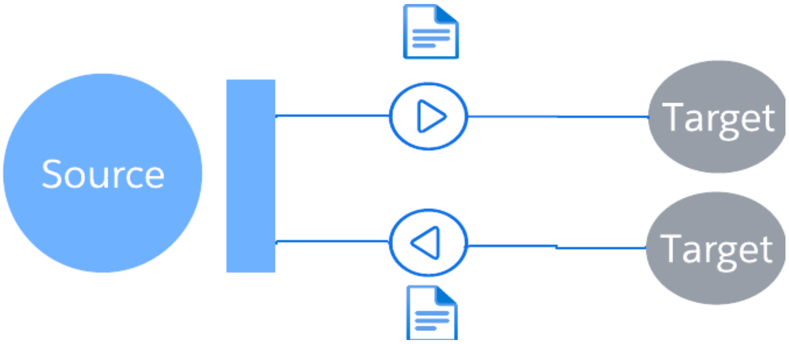


Fig – API-led example for an Async pattern

Considerations during decision making process for tool/pattern selection:

Source push	The Source system should have the ability to push information
Reliability	Build durable or reliable message delivery within the MQ level for guaranteed delivery of messages
Scalability	Scalability of integration is possible by scaling the MQ level
Private vs Public MQ	Consider deploying separate messaging infrastructure for the internal applications and external public applications

Publish Subscribe



Use-case scenarios:

The pattern should be used in following use-case scenarios:

- Event Broadcast
- Multiple Systems Consumption
- Alternative to Shared Datastore
- Durable and Reliable Operations

Example: Workday ⍰ SuccessFactors and EDMS

When NOT to use:

Do not use this pattern in following cases:

- On Demand operations with sequential dependencies
- Time-sensitive synchronous operations

API-led approach and guidelines:

- Key to utilize an Async pattern is deployment of a messaging system. – such as IBM MQ, Active MQ, AnyPoint MQ. Topics within MQ are created to which multiple consumers can subscribe.
- A System API is developed in front of the systems that generate/consume data.

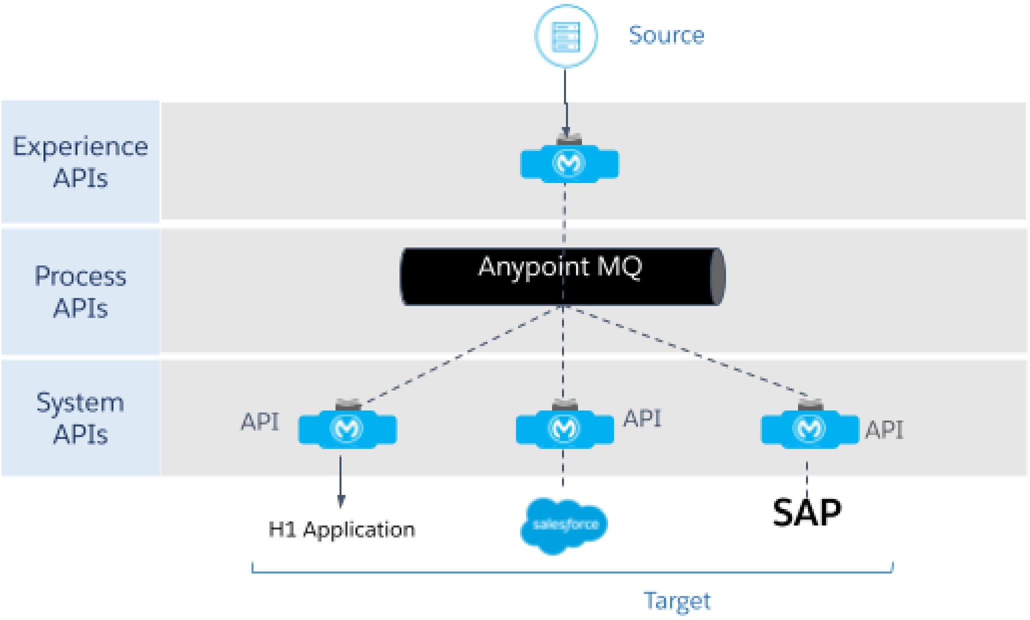
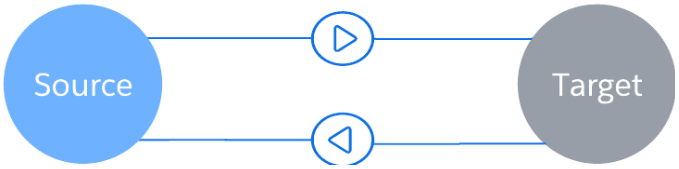


Fig – API-led example for a Pub-Sub pattern

Considerations during decision making process for tool/pattern selection:

Data format	Ensure a common messaging data language is developed (preferably on Domain Driven Design concepts)
Durability	To ensure durability and guaranteed delivery, appropriate configurations must be made within the MQ
Security	Consider deploying separate internal and external MQ for the internal and external consumers
Reusability	This pattern allows for maximum reusability across the consumers
Fault Tolerant	This pattern provides increased fault tolerance with appropriate decoupling and durability

Bi-Directional Sync



Use-case scenarios:

The pattern should be used in following use-case scenarios:

- Keep information in source and target systems in sync.
- Data is mastered in Target but used in source frequently on a real-time basis.
- Data changes fairly frequently in either system

  Example: Capser to SAP for order allocation

When NOT to use:

Do not use this pattern in following cases:

- When there is a significant performance impact on the system due to the overhead of constantly synchronizing data bi-directionally.
- When the system has complex data structures and dependencies that may make it difficult to manage bi-directional synchronization.
- When there are potential conflicts in data updates that cannot be easily resolved, leading to data inconsistencies.
- When the system requires a high degree of scalability, and the overhead of bi-directional synchronization may limit scalability.
- When the system has security or compliance requirements that require strict controls over data access, bi-directional synchronization may introduce potential security risks.
- When there are differences in the data models or schema between the systems that are being synchronized, making bi-directional synchronization difficult or impossible.
- When the systems being synchronized have different data retention policies, making it difficult to reconcile differences in data that may be purged from one system.
- When the cost of implementing and maintaining bi-directional synchronization outweighs the benefits it provides.

API-led approach and guidelines:

- For this particular use case, experience APIs are not necessary as its bi-directional sync between two systems.

- At the process layer, one can develop a mule application that orchestrates the synchronisation of data between the backend systems using the Batch component. This batch process can be triggered by the Scheduler component, so it runs a regular basis. Within the batch scope, the data to be synced from Source system is retrieved, validated and transformed into the Target System. The data is then persisted into the Source and core systems of record by calling the relevant system APIs.

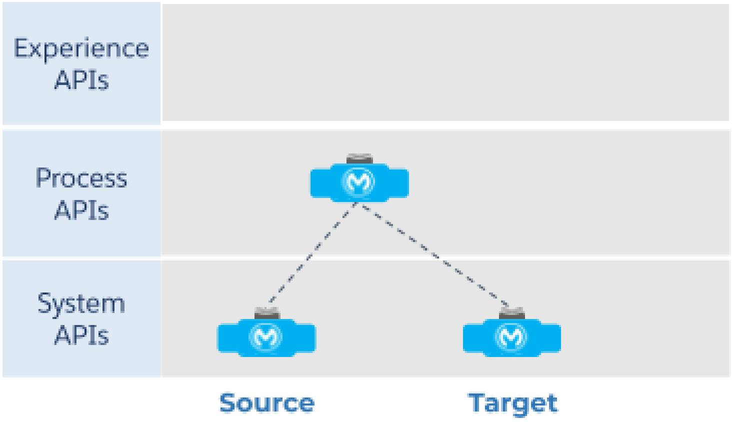


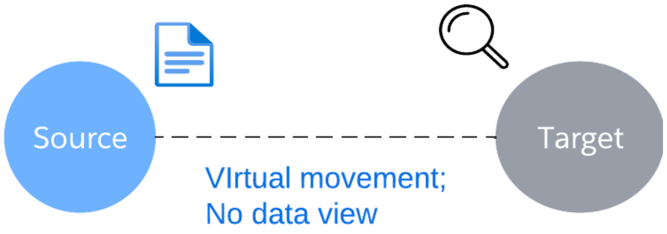
Fig – API-led example for a bi-directional sync pattern

Considerations during decision making process for tool/pattern selection:

Data consistency	One of the primary concerns when implementing bi-directional synchronization is ensuring that data remains consistent between the two systems. This requires careful planning and management of data conflicts, as well as establishing clear rules for how changes will be propagated between the systems.
Performance	Bi-directional synchronization can impose a significant performance overhead on both systems, especially when dealing with large volumes of data. Careful attention must be paid to optimizing synchronization algorithms and minimizing unnecessary data transfers.
Scalability	Bi-directional synchronization can become increasingly complex and resource intensive as the number of systems involved grows. Careful planning and architecture are required to ensure that the system can scale effectively to meet increasing demands.
Complexity	Bi-directional synchronization can be complex to implement and manage, particularly when dealing with multiple systems with different data models and schemas. Careful planning and management are required to ensure that synchronization remains reliable and efficient.
Data retention	Bi-directional synchronization requires careful management of data retention policies, as changes made in one system may need to be propagated to others even if the data has been purged from the original system.
Availability	Bi-directional synchronization requires both systems to be available and accessible to work effectively. Redundancy and failover mechanisms may be required to ensure that synchronization can continue in the event of system failures or downtime.

©MuleSoft, LLC 2020

Virtualization



Use-case scenarios:

The pattern should be used in following use-case scenarios:

- Federations of data from many data sources
- Complex joins across multiple sources
- Process large volume of records
- Complex processing of large payload sizes
- Strict SLAs where network hops, data processing time are critical.
- Data cannot be copied due to Security or Governance

When NOT to use:

Do not use this pattern in following cases:

- Physical copies of data is required
- Different security requirements within consuming application

API-led approach and guidelines:

When deploying a data virtualization solution in the integration ecosystem, following options could be used:

- Option 1: Develop a *proxy* MuleSoft API layer either using Flex Gateway or a MuleSoft proxy and treat the Data Virtualization platform yet another source of the data. In this option, the MuleSoft hosted proxy API acts as a pass-through but with ability to control and manage the traffic.
- Option 2: If the Data Virtualization platform can expose a HTTP/S endpoint for the data assets, these endpoints could be directly called by a higher-level process or experience API.

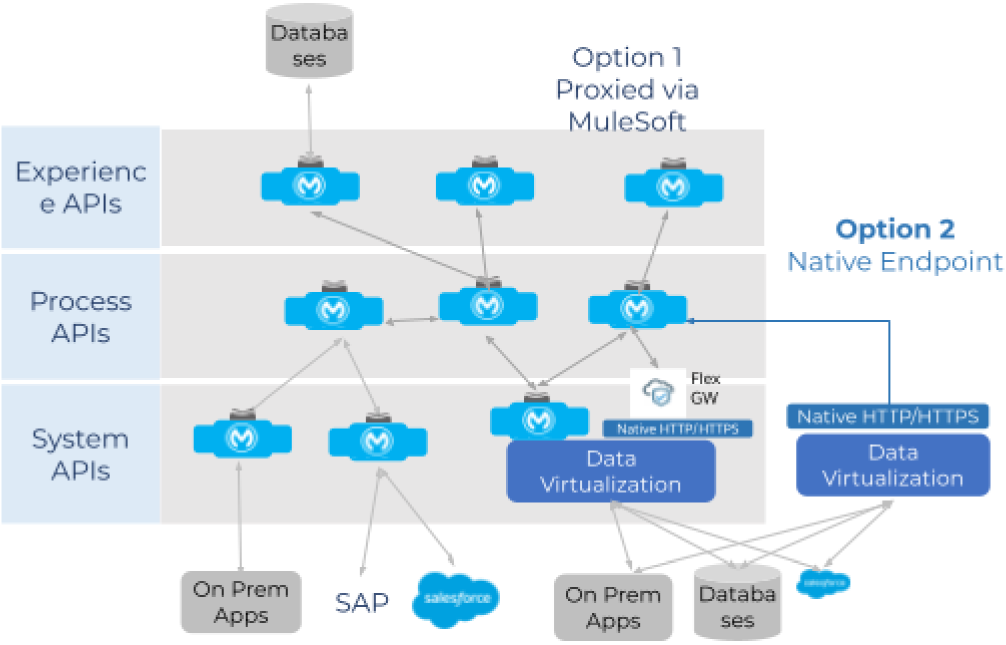


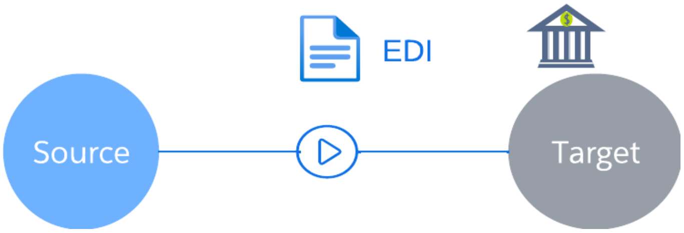
Fig – API-led options for a data virtualisation pattern

Considerations during decision making process for tool/pattern selection:

Payload Size	When choosing MuleSoft vs Data Virtualization, consider the payload size. If a payload size > 10 MB, consider deploying the data virtualization platform
Single/Multi Source	Consider deploying Data Virtualization platform when there are multiple sources of data that are interrelated and require complex joins, data deduplication etc. before reaching the endpoint
Timing/Latency	Many virtualization platforms provide caching abilities. Consider deploying caching features in conjunction with the data freshness requirements
Security	When exposing the data assets from Data Virtualization platform, consider utilizing the API manager layer for better access, throttling and security for the data
Standards and interoperability	Consider the extent to which the data sources being integrated are standardized and interoperable. Data virtualization works best when data sources are standardized and can be easily mapped to a common data model.
Availability	Consider the availability requirements for the integrated data, including redundancy and failover mechanisms. Data virtualization can introduce additional points of failure, so it is important to ensure that the system remains available even in the event of failures.

Scalability	Consider the scalability requirements for the integrated data, including the ability to handle increasing data volumes and new data sources. Data virtualization can struggle to scale to handle large data volumes or complex data structures, so it is important to plan for scalability from the outset.

B2B Integration



Use-case scenarios:

The pattern should be used in following use-case scenarios:

- Customers/ External Partner / Third Party Requirements
- Data Formats / Protocols / Functionality Only Supported by a B2B Provider
- Legacy systems limitation

Example: SAP to/from Opentext to customer

When NOT to use:

Do not use this pattern in following cases:

Low-volume integrations	If there is only a need to integrate with a small number of partners or have low volumes of data exchange, the B2B integration pattern may be overkill. In these situations, simpler integration patterns may be more appropriate, such as direct API-to-API integrations.
Non-standard B2B formats	If the partners use non-standard B2B formats that cannot be easily transformed into APIs, the B2B integration pattern may not be the best approach. In these situations, specialized tools or custom development may be required to transform the data into a format that can be consumed by the systems.
Legacy systems	If the partners are using legacy systems that do not support modern B2B protocols or formats, the B2B integration pattern may not be the best approach. In these situations, specialized connectors or adapters may be required to enable integration with these systems.

Tight coupling with partners	If the integration requires tight coupling with the partners, such as sharing database connections or direct access to partner systems, the B2B integration pattern may not be the best approach. In these situations, direct point-to-point integrations may be more appropriate.
Resource constraints	If there is limited resources or budget for implementing integrations, the B2B integration pattern may not be the most cost-effective approach. In such situations, simpler integration patterns or outsourcing to a third-party provider may be more appropriate.

API-led approach and guidelines:

- API-led approach to file transfers, promotes reuse of the process and system APIs for multiple source systems, partners and integrations into third party systems.
- The B2B Transforms are to be treated as experience level APIs that interact with the external partner and organizations.
- If the partners use EDI or other B2B formats, the B2B integration pattern can be used to transform these formats into APIs that can be consumed by <<customer>> systems. This can be done using MuleSoft's data mapping and transformation capabilities, which can convert EDI or other B2B formats into JSON or XML.
- If there is a need to integrate with multiple parties, such as suppliers, customers, and logistics providers, the B2B integration pattern can be used to orchestrate these integrations. This can be done using MuleSoft's API-led approach, which enables you to create reusable APIs and connectors that can be easily integrated with partner systems.

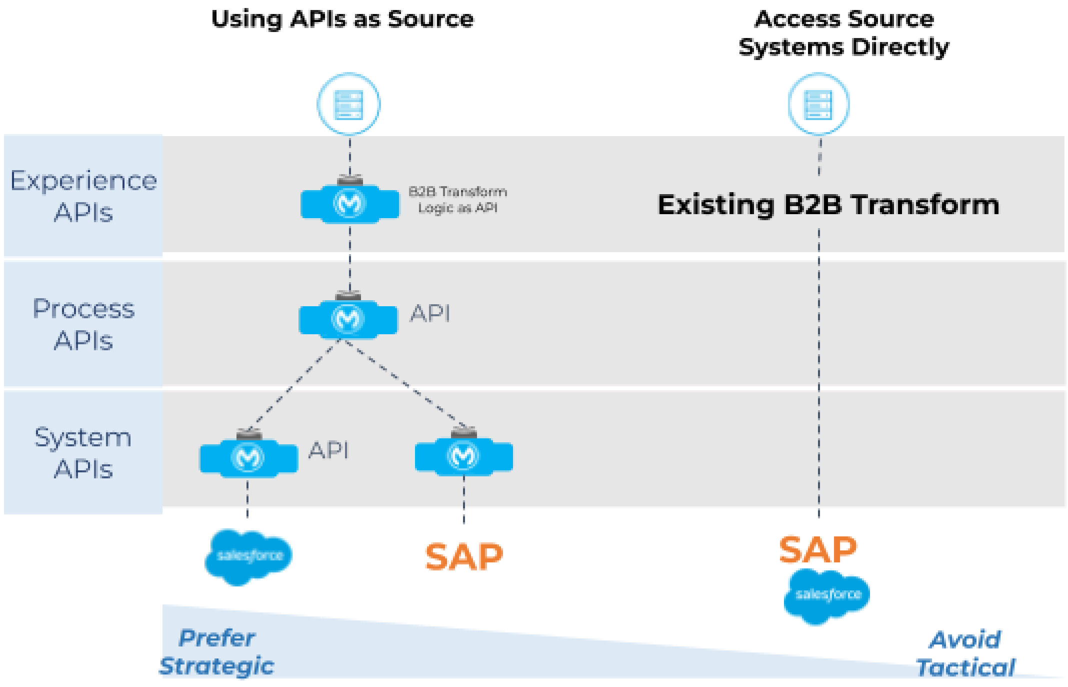
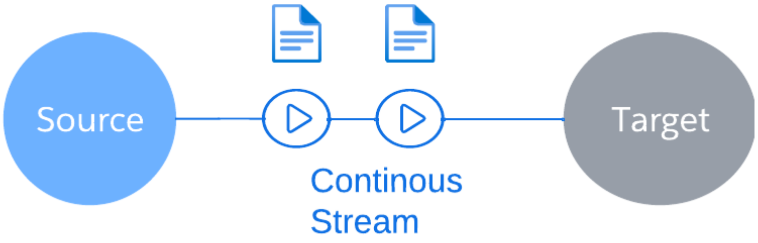


Fig – API-led options for a B2B pattern

Considerations during decision making process for tool/pattern selection:

Security	B2B integrations typically involve the exchange of sensitive data between organizations, so security is a critical consideration. Encryption, authentication, and access control should be implemented to ensure that data is exchanged securely, and only authorized users have access to it. Some B2B customers can specify additional security measures that are not part of standard security requirement.
Scalability	B2B integrations often involve large volumes of data and complex workflows, so the system should be designed to handle high traffic volumes and scale as the business grows.
Standards and compliance	There are various industry and regulatory standards that must be followed when exchanging data, such as EDI (Electronic Data Interchange) and HIPAA (Health Insurance Portability and Accountability Act). The B2B integration system should be designed to comply with these standards. When choosing MuleSoft APIs vs. existing integrations, consider the protocol supported by the recipient and its implementation using MuleSoft.
Connectivity	B2B integrations involve connecting with external partners, suppliers, and customers, so the system should be designed to support different connectivity protocols, such as FTP, SFTP, AS2, and web services.
Monitoring and error handling	The B2B integration system should include monitoring and error handling capabilities to ensure that data is exchanged correctly, and any errors are detected and resolved quickly.
Data mapping and transformation	The B2B integration system should be able to map and transform data between different systems and formats, such as XML, JSON, and CSV, to ensure that data is exchanged correctly.
Business continuity	B2B integrations are critical to business operations, so the system should be designed to ensure high availability and business continuity, such as implementing failover and disaster recovery measures.

Streaming



Use-case scenarios:

The pattern should be used in following use-case scenarios:

- Real time data access especially in scenarios such as data analysis, real-time dashboarding, log analysis etc. which would enable real-time decision making, predictive analytics and automated actions.
- IoT(Internet of things) devices generate vast amounts of data in real-time, such as sensor readings, telemetry data, and machine logs. The streaming integration pattern can be used to ingest, process, and analyze this data in real-time to enable predictive maintenance, real-time monitoring, and automated decision-making
- Large amount of data processing/syncing is needed between systems

When NOT to use:

Do not use this pattern in following cases:

Adhoc/batch generation	Adhoc/batch data generation is not a great fit for this pattern.
Low volume of data	If the volume of data is low and real-time processing is not critical, batch processing may be a more efficient approach. In this case, the data can be processed in batches rather than streamed in real-time.
High latency	The streaming integration pattern is designed for low-latency processing, but there may be situations where high latency is acceptable. For example, if the data is not time-sensitive, batch processing may be a more cost-effective option.
Limited resources	The streaming integration pattern requires significant resources, such as computing power and memory, to support real-time data processing. If resources are limited, batch processing may be a more feasible option.
Complexity	The streaming integration pattern can be complex to implement and maintain, particularly if the data comes from multiple sources or if there are multiple processing steps involved. If the integration is too complex, it may be more efficient to use another integration pattern.

API-led approach and guidelines:

Define clear API contracts	When using the streaming integration pattern in an API-led architecture, it is important to define clear API contracts that define the data format, structure, and expected behavior of the API. This will ensure that data can be efficiently processed and exchanged between different systems
Implement event-driven architecture	An event-driven architecture can be used to facilitate real-time processing of streaming data. This involves using events to trigger actions in real-time, rather than relying on batch processing.
Leverage streaming connectors	Streaming connectors can be used to connect to streaming data sources, such as Apache Kafka or Apache Pulsar, and to efficiently process streaming data. These connectors can also be used to integrate with other systems, such as databases or APIs.
Use message queues	Message queues can be used to buffer and store streaming data, providing a mechanism for fault-tolerant processing and ensuring that data is not lost in the event of a failure.

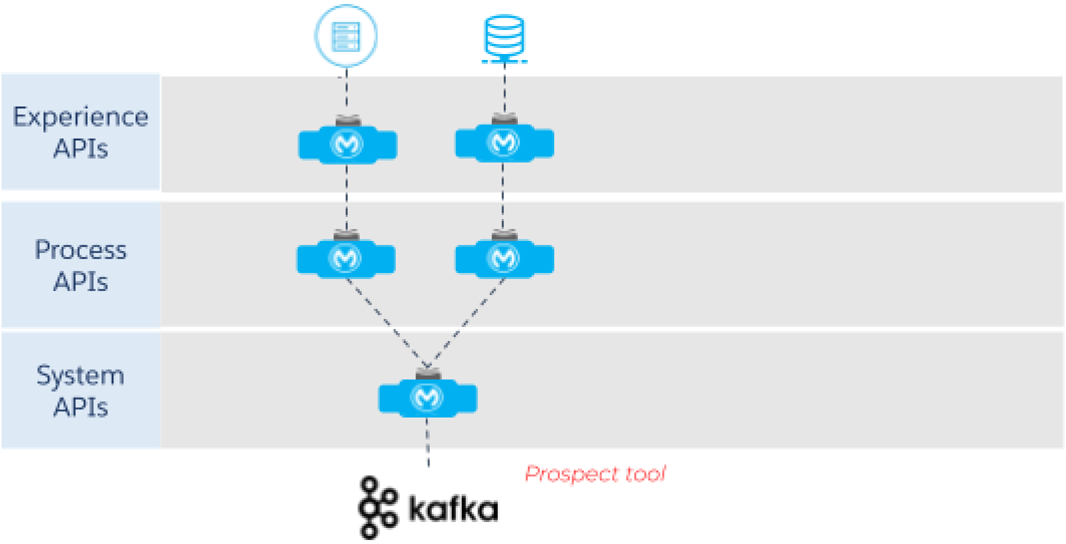


Fig – API-led for a streaming pattern

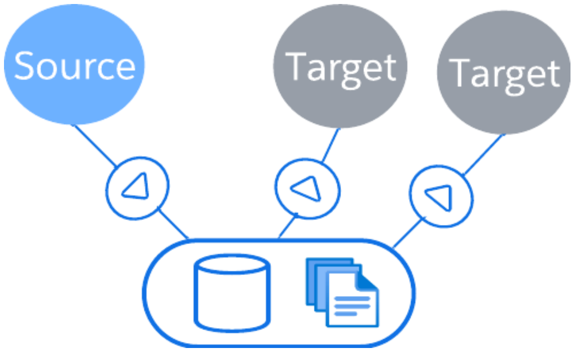
Considerations during decision making process for tool/pattern selection:

Data format	Consider the payload size of streaming data
Technology/Skills	Considering no existing streaming platform and teams at <<customer>>, it would require significant investment

©MuleSoft, LLC 2020

Latency	Batch movement might be an alternative is source can store and transmit in batches
Scalability	Increase in streaming data volume will require horizontal scaling of integration processors
Durability	Additional considerations are required if message durability is needed

Shared data store



Use-case scenarios:

Microservices architecture	In a microservices architecture, each service may have its own data store. However, in some cases, multiple services may need to access the same data, such as customer information or order history. In this case, a shared data store can be used to centralize the data and make it available to all the services that need it.
Multi-channel applications	Multi-channel applications, such as e-commerce websites or mobile applications, may need to access the same data across different channels. For example, a customer may add an item to their shopping cart on their mobile device, and then later view the same shopping cart on a desktop computer. In this case, a shared data store can be used to keep the shopping cart data in sync across different channels.
Analytics and reporting	Analytics and reporting applications often need to access data from multiple sources. In this case, a shared data store can be used to consolidate the data and make it available to the analytics and reporting tools.
Legacy system integration	When integrating with legacy systems, a shared data store can be used to bridge the gap between the modern and legacy systems. The shared data store can be used to store and share data between the modern and legacy systems, allowing them to work together more seamlessly.

Collaboration and workflow management:	Collaboration and workflow management applications may require access to shared data, such as project plans or team calendars. A shared data store can be used to store this data and make it available to all the users who need it.

When NOT to use:

Do not use this pattern in following cases:

Highly Sensitive Data	When data is highly sensitive or requires strict access controls, shared data stores might pose risks.
Real-Time Requirements	Situations where immediate data access or real-time updates are crucial might not align with this pattern.
Data Governance and Compliance	Compliance standards requiring distinct data ownership or segregation might conflict with shared data stores.
Complexity and Scalability	Complex data models or scalability issues due to the volume of shared data might make this pattern impractical.
Data Consistency Challenges	Maintaining data consistency across systems with conflicting models or formats can be challenging.
Cost and Maintenance	If the cost and effort of maintaining a shared data store outweigh the benefits for smaller datasets or limited sharing requirements.
Performance Implications	Instances where a shared data store might introduce performance bottlenecks or contention issues.

API-led approach and guidelines:

This integration pattern requires very strong data governance on shared schema. In addition, careful consideration must be made for consistency and transactionality of the data.

Use this integration pattern when source to shared data store integration already exists. API-led approach can be used to integrate consumers with shared datastore.

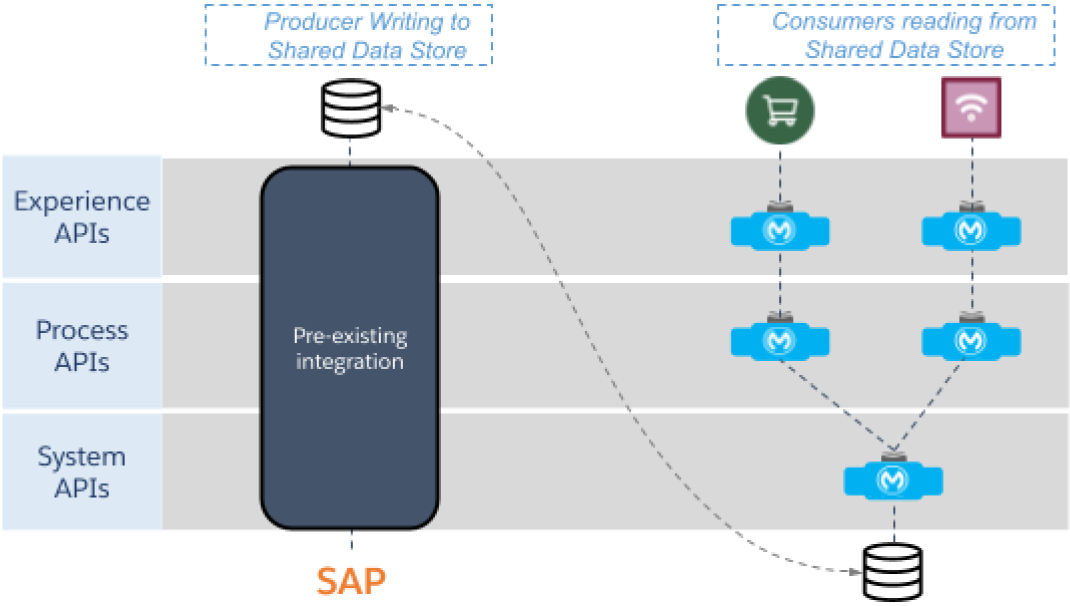

Producer Writing to Shared Data Store
Consumers reading from Shared Data Store
Experience APIs
Process APIs
System APIs
Pre-existing integration
SAP

Fig – API-led for a shared data store pattern

Considerations during decision making process for tool/pattern selection:

Compatibility and Integration	Ensure compatibility and ease of integration with existing systems and applications.
Data Consistency and Synchronization	Consider the need for real-time data consistency and synchronization across multiple systems.
Scalability and Performance	Evaluate the scalability and performance requirements concerning data volume and user access.
Operational Overheads and Maintenance	Consider the operational complexities, configuration ease, and ongoing maintenance requirements.
Data Security and Access Control	Prioritize robust security measures and access controls to protect sensitive shared data.

About MuleSoft, a Salesforce company

MuleSoft, provider of the world’s #1 integration and API platform, makes it easy to connect data from any system – no matter where it resides – to create connected experiences, faster. Thousands of organizations across industries rely on MuleSoft to realize speed, agility and innovation at scale. For more information, visit https://www.mulesoft.com.

MuleSoft is a registered trademark of MuleSoft, LLC, a Salesforce company. All other marks are those of respective owners.

©MuleSoft, LLC 2020