Mule Design Guidelines

1 Anti-Patterns

Area	**Anti-Patterns**	**Solution**
Tight Coupling	Developing multiple APIs in a single Mule Project	1 API : 1 Mule app (there are exceptions)
ETL	Using Mule for ETL jobs, and expect ETL traditional features (e.g. staging, monitoring, etc)	Though Mule supports ETL processing via Batch components, it's not an ETL tool with all the traditional features. Use dedicated ETL tools for large volumes of data (> 1 GB) and when specific ETL's features are needed
Business Logic & Rules	Implementing Business Logic & Rules in Mule expecting features from a BR engine (e.g. rules hot-deployment, testing scenarios, etc)	Use dedicated BRMS tools. Get business rules from Mule apps via APIs (e.g. Drools APIs)
Business Process Management	Using Mule as a BPM tool	Use BPM for such needs

2 Synchronous

- API (Restful Services) developed using RAML for all synchronous communication
- API proxy used to proxy existing web services (SOAP/REST)
- Use a Message Broker/VM for Async needs inside synchronous scenarios

3 Asynchronous

- Mule support Asynchronous patterns via “Async” component, Virtual Machine (VM) and Java Message Service (JMS) connectors
- VM component can be used to convert non-transactional components like HTTP, FTP, TCP into transaction component which helps in rolling back transactions, retry mechanisms etc.
- Asynchronous message handling is one of the keys to use Mule effectively. Mule support Asynchronous processing patterns in many different ways like
- (only Mule 3.x) Setting exchange-pattern of a message source to “one-way” enables asynchronous processing for a flow. For Mule 4.x make use of the Async scope.
- Asynchronous processing for a flow can be tuned by defining a queued-asynchronous-processing-strategy. Multiple queued-asynchronous-processing-strategy can be defined and set using the flow’s “processingStrategy” attributes

4 Publish/Subscribe

- A Message Broker (ActiveMQ, AnypointMQ, Kafka, RabbitMQ, WMQ, etc) should be used for Asynchronous integrations to implement pub-sub model
- Create topics based on application needs, topics can be classified by application, message types etc.

5 File Handling

- File handling should always ensure zero file/message loss and should include a reprocessing strategy or backup in case of error files
- Do not delete or overwrite input/output files
- Be careful when using the **autoDelete** and the **moveToDirectory** attributes of the **FileTransport**
- If streaming is enabled for each file that is processed, make sure that the stream is properly close after reading
- Always ensure that the file/ message has been archived correctly once entered in the mule application / flow
- Don't save file contents / complete file / payload in session unless it's required
- Define correct file handling strategies, considering all the failure / success / transaction / unhandled error scenarios
- Ensure that the directories used in the file handling have correct read / write permissions
- When a file reading process is scheduled, ensure that the same file is not picked up for processing when the next schedule starts.
- Create directories like “**working**”, “**error**”, “**processed**” directories to manage files. For example, when a file is selected for processing, move it to the “working” directory, on successful processing move it to the “processed” directory, if the process fails, move it to the "error” directory

6 Large Data Handling

- Mule has out of the box Batch Processing components that can be used for Data synchronization between systems for large data
    - Reference: https://docs.mulesoft.com/mule-user-guide/v/3.9/batch-processing
- Data could be considered large in Mule if it's > 20 MB (per request/transaction). It also depends on worker size, complexity of implementation etc.
- Dataweave, Streaming, etc can be used appropriately based on the use-case

7 Error Handling

Consider the following categories of errors when designing Mule flows:

- **Business Logic errors:** Thrown by validations and conditions based on the use case requirements.
- **Systems' availability errors:** Errors based on the source/target systems' availability and re-connection strategies (Connection timeout, max retries reached, etc)

Don't include error handling for the following categories:

- **Development errors:** To cover bad development practices, like bad parsing, bad transformation inputs, etc.

8 Batch Processing

Original Author: Paul Anderson

8.1 Introduction

The reasons batch-processing is necessary originate with the need to process accumulated workload in the form of batch-files, polled database reads or polled web-service requests.

Modern architecture favors real-time processing where possible, for many reasons. However, if batch-processing is currently a real need for businesses, this section details best-practices for doing this.

The Mule batch-processing module is documented in detail here:

https://docs.mulesoft.com/mule-user-guide/v/3.9//batch-processing

8.2 Considerations for effective batch-processing

8.2.1 Batch Size

Between the extremes of a batch-size of one or batch-size of (all available records) lies an optimum point. There is no a-priori rule which defines the optimum batch-size; this needs to be determined by tuning and experimentation. Regardless of the batch-implementation it is recommended that the batch-size be an externalized parameter to ease tuning and maintenance.

Note that in the case of file-processing, batch-size should not automatically be considered to be the size of the file. Commonsense and scientific assessment should drive the setting of batch-size.

8.2.2 Error Protocol

Errors during batch-processing need to be handled in one of two ways:

a. an error invalidates the whole batch - rollback is necessary
b. individual errors can be processed tolerated, allowing the rest of the records to process to completion.

In real-life there are degrees of error-tolerance, and Mule Batch component supports these modes:

i. a single error terminates the batch
ii. an error-threshold is tolerated.

In either case above, it is the user's responsibility to determine and implement roll-back actions (since these are too general to code directly into a generally-reusable component. The Mule batch component documentation listed above gives all the details necessary

8.2.3 Completion Actions

On completion of a batch, it is usually necessary to notify a system or job-function, possibly including generating and dispatching a report. Mule Batch component supports these actions - see Batch Complete section of the batch-processor.

9 Saleforce to Data Hub Integration using Streaming API

9.1 Introduction

Often there is a need to synchronize data from Salesforce in to Customer's data warehouse for reporting and even for transactional purposes.
Saleforce provides so many flavors of streaming events to publish changes and Mule Salesforce connector support these streaming API.
This article provides the option of which streaming events to choose and appropriate mule design.
Salesforce provides 3 streaming options:

- Platform Events - custom object with defined structure, both internal and external users can publish.
- Push Topics - change event capture of busines objects. Supports all custom objects but only few standard objects
- Generic Events - Events that create string notifications/publishing.

While this article address which streaming option to use from Salesforce, it has respective Mule design goal as well. Following are Mule design goals:

- Create common framework to publish these events to on-premise data hub.
- Provide near real-time and reliable data synchronization with proper exception handling and replay features.
- Avoid creating new integration flows, when more and more business object are added for synchronization.

9.2 Audience

Technical Architects, Developers

9.3 Best Practices

- Due to Salesforce API usage(Number of Transactions per day) limitations and number of business objects that will require synchronization to datahub, it's not a good practice to
- create separate integration flows for each business object.
- Push Topic events has limitation on data size and doesn't support large text attributes. All custom objects and only a subset (around 10) of standard objets are supported.
- Generic events are meant for publishing string messages rather than objects.
- Platform Events don't have any of the above limitations, can be used as one custom generic event to publish all business object changes.
- Create one integration flow using SFDC Connector(Streaming) and route to different backend kafka queues or endpoints using business object name in the event
- Add exception handler using another MQ solution like Active MQ or Anypoint MQ to store error events for replay as the default replay from Salesforce is only 24 hours.

- Refer the **how-to section** on more details on the implementation.

Salesforce documentation on which streaming events to use and limitations within each streaming event.

Which streaming to Use: https://developer.salesforce.com/blogs/2018/07/which-streaming-event-do-i-use.html
Platform Event Allocations: https://developer.salesforce.com/docs/atlas.en-us.214.0.platform_events.meta/platform_events/platform_event_limits.htm
Push topic Events Allocations: https://developer.salesforce.com/docs/atlas.en-us.api_streaming.meta/api_streaming/limits.htm
Generic Event Streaming Allocations: https://developer.salesforce.com/docs/atlas.en-us.api_streaming.meta/api_streaming/limits_generic.htm