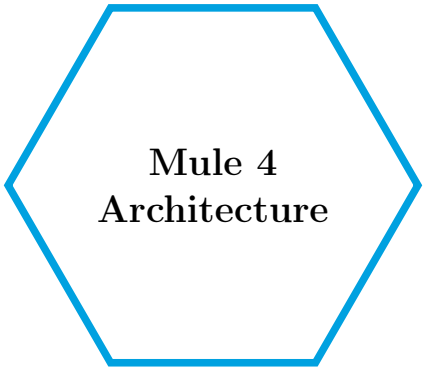
MuleSoft Advanced Mastery

Architecture, DataWeave & Error Handling



API-Led Connectivity

Topics Covered:

API-Led Architecture (XAPI-PAPI-SAPI) · DataWeave 2.0
Autodiscovery · Flow Types · Error Handling

Prepared for:
MuleSoft Developer
December 9, 2025

Contents

Chapter 1

Architecture: API-Led Connectivity

1.1 The Three-Layered Approach

MuleSoft advocates for **API-Led Connectivity**, a methodological way to connect data to applications through reusable and purposeful APIs.

1.1.1 The Layers

1. **Experience Layer (XAPI):**
    - *Purpose:* Formats data for specific end-users (Mobile, Web, B2B).
    - *Why:* Decouples the frontend from backend complexity.
2. **Process Layer (PAPI):**
    - *Purpose:* Orchestration, aggregation, and business logic.
    - *Why:* Eliminates silos by combining data from multiple systems.
3. **System Layer (SAPI):**
    - *Purpose:* Unlocks raw data from backend systems (SAP, Salesforce, DB).
    - *Why:* Protects the backend from direct access and changes.

1.2 Architecture Diagram

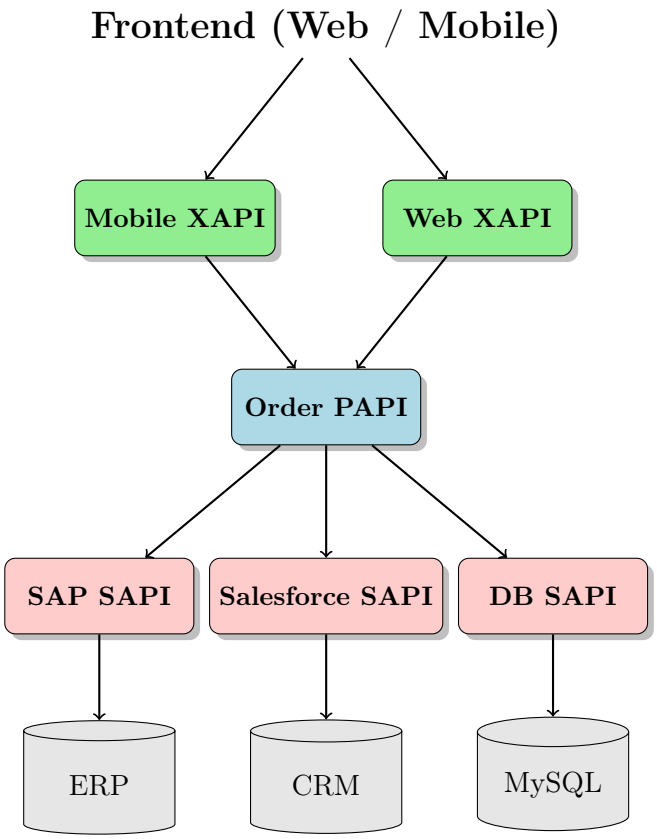


Figure 1.1: Frontend to Backend Flow (API-Led Connectivity)

1.3 Real-Time Use Case: E-Commerce Order

- **Frontend:** User clicks "Buy" on the iPhone App.
- **XAPI:** `mobile-exp-api` receives JSON.
- **PAPI:** `order-process-api` receives the order. It calls `customer-sapi` to validate the user and `inventory-sapi` to check stock.
- **SAPI:** `sap-sapi` actually posts the invoice to the SAP ERP system.

Chapter 2

Mule Flows: Main, Private & Sub-Flows

2.1 Flow Types Definitions

Type	Source	Error Handling	Threading
Main Flow	Has a Source (e.g., HTTP Listener, Scheduler).	Has its own Error Handling block.	Can be Async or Sync.
Private Flow	No Source. Called via `Flow Ref`.	Has its own Error Handling block.	Runs in the same thread (mostly).
Sub-Flow	No Source. Called via `Flow Ref`.	**NO** Error Handling (Inherits from parent).	strictly Synchronous (same thread).

Table 2.1: Comparison of Flow Types

2.2 Internal Working Usage

2.2.1 Why use Private Flows over Sub-Flows?

Use a **Private Flow** when you need specific error handling for a segment of logic (e.g., "If this specific database call fails, don't stop the whole flow, just log it and continue").

2.2.2 Why use Sub-Flows?

Use a **Sub-Flow** purely for code reusability and modularity where the parent flow should handle any errors. It is lighter on memory.

```
<flow name="mainFlow">
    <http:listener path="/start"/>
    <flow-ref name="mySubFlow"/>
    <flow-ref name="myPrivateFlow"/>
</flow>

<sub-flow name="mySubFlow">
    <logger message="I am a subflow"/>
</sub-flow>

<flow name="myPrivateFlow">
    <logger message="I am a private flow"/>
    <error-handler>
```

```
14          <on-error-continue>
15              <logger message="Handled internally"/>
16          </on-error-continue>
17      </error-handler>
18  </flow>
```

Listing 2.1: Private vs Sub Flow XML

Chapter 3

DataWeave 2.0

3.1 Definition Purpose

DataWeave is the functional programming language used by MuleSoft for data transformation. It transforms data from one format (JSON, XML, CSV, Java) to another.

3.2 Core Functions Examples

3.2.1 The ‘map‘ Operator

Used to iterate over arrays.

Input (JSON): ‘["id":1, "name":"A", "id":2, "name":"B"]‘

Script:

```
%dw 2.0
output application/xml
---
users: {
    (payload map (item, index) -> {
        user: {
            id: item.id,
            userName: item.name,
            position: index
        }
    })
}
```

3.2.2 The ‘filter‘ Operator

Filters an array based on a condition.

```
%dw 2.0
output application/json
---
payload filter ($.age > 18)
```

3.2.3 Advanced: ‘reduce‘

Reduces an array to a single value (e.g., sum, concatenation).

```
%dw 2.0
output application/json
var nums = [1, 2, 3, 4]
---
```

```
{
    total: nums reduce (item, accumulator = 0) -> item + accumulator
}
// Output: { "total": 10 }
```

DataWeave Performance

Avoid transforming massive payloads (100MB+) in-memory. For large files, use **Streaming** mode (‘deferred=true‘) to process data without loading it all into RAM.

Chapter 4

API Autodiscovery

4.1 Definition

API Autodiscovery is the mechanism that links a Mule application running in the Runtime Plane (CloudHub) to an API configured in the Control Plane (API Manager).

4.2 Why is it used?

It allows you to apply **Policies** (Rate Limiting, Client ID Enforcement, IP Whitelisting) *without* changing the application code.

4.3 How it works (The Gatekeeper)

When the Mule App starts:

1. The runtime reads the `apiId` and `client_id/secret`.
2. It calls the Anypoint Platform to download policies associated with that ID.
3. **Gatekeeper Mode:** The API remains "blocked" (returns 503) until the policies are successfully applied.

4.4 Configuration Guidelines

1. **Global Element:** Add ‘API Autodiscovery‘ in ‘global.xml‘.
2. **Flow Reference:** Point it to the main flow (where the HTTP Listener is).
3. **Properties:**
   - `apiId`: From API Manager.
   - `flowRef`: The name of your main flow.

```
<api-gateway:autodiscovery
    apiId="${api.id}"
    ignoreBasePath="true"
    doc:name="API Autodiscovery"
    flowRef="my-main-api-flow" />
```

Listing 4.1: Autodiscovery Config

Chapter 5

Error Handling

5.1 On Error Propagate vs. Continue

On Error Propagate	On Error Continue
Rethrows the error. The transaction is marked as FAILED. Returns HTTP 500 (by default). Used when the parent flow MUST know something went wrong.	**Swallows** the error. The transaction is marked as SUCCESS. Returns HTTP 200 (by default). Used when you want to handle the error gracefully and send a custom response.

5.2 Scenario: PAPI calling SAPI

Imagine an Order PAPI calls a Customer SAPI.

1. **Scenario A (Propagate):** SAPI is down. PAPI's HTTP Request throws an error. Inside PAPI, we use *On Error Propagate*.
    - *Result:* PAPI stops, sends HTTP 500 to the client (Mobile App).
2. **Scenario B (Continue):** SAPI is down. Inside PAPI, we use *On Error Continue*. We set the payload to ' "message": "Try again later" '.
    - *Result:* PAPI sends HTTP 200 OK to the client with the custom message. The client thinks it succeeded.

Common Mistake

Putting 'On Error Continue' in the **Main Flow** creates "False Positives". The monitoring tools will show green (Success) even though the business logic failed. Use 'Propagate' in main flows usually, and 'Continue' inside transformations or optional calls.

Chapter 6

Interview Guide

6.1 Top Interview Questions

6.1.1 Q1: What is the Gatekeeper in Autodiscovery?

Answer: It is a security mechanism that prevents the API from processing requests until all policies (like OAuth, Rate Limiting) are successfully downloaded and applied from the API Manager.

6.1.2 Q2: Can a Sub-Flow have exception handling?

Answer: No. A sub-flow effectively processes as if its processors were inside the calling flow. It shares the exception strategy of the caller.

6.1.3 Q3: Explain 'lookup' function in DataWeave.

Answer: It allows a DataWeave script to call a flow in the Mule application and retrieve the result. Useful for complex lookups during transformation.

6.2 Summary

- **Architecture:** Always think in layers (System, Process, Experience) to promote reuse.
- **DataWeave:** Is powerful. Use 'map' for arrays, 'reduce' for aggregation.
- **Flows:** Private flows for error isolation; Sub-flows for reuse.
- **Autodiscovery:** The bridge between code and governance.