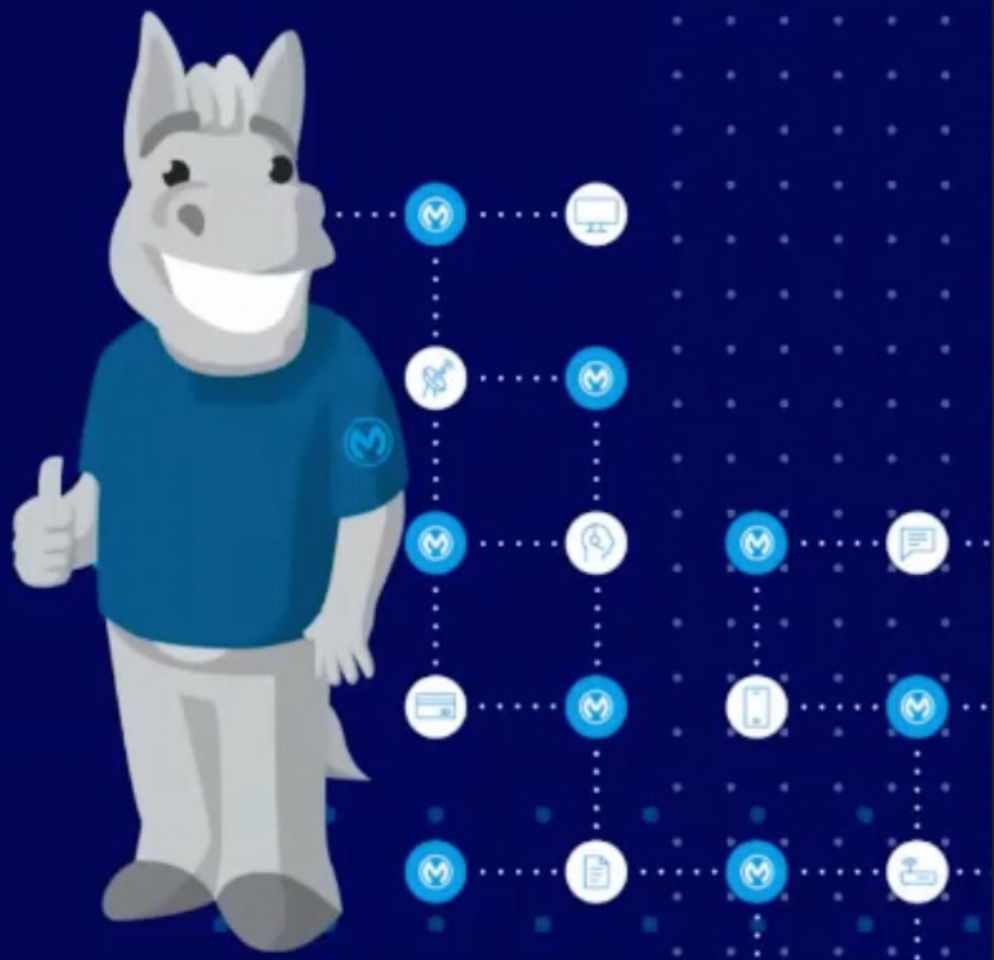
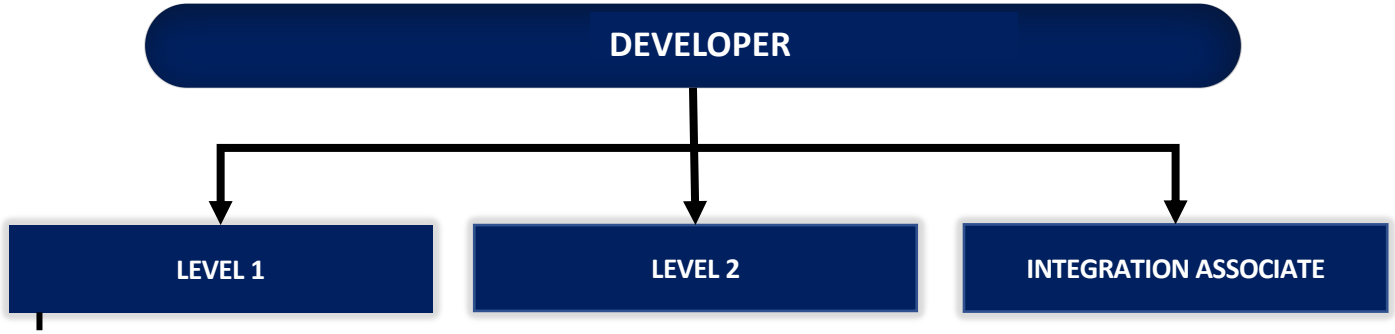
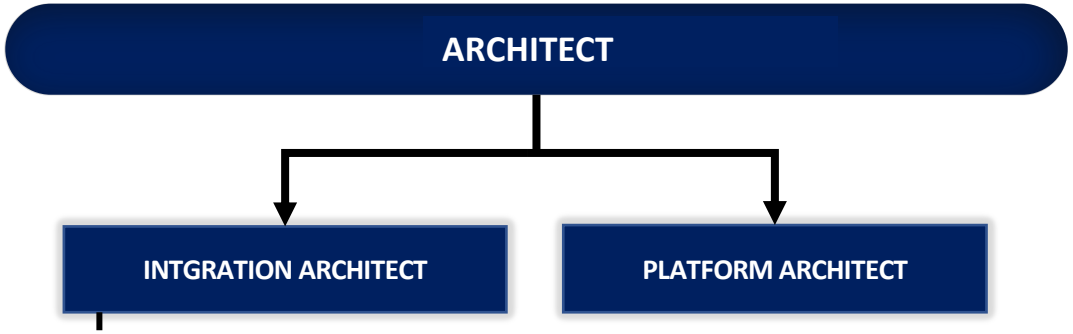
Chinna’s MuleSoft course



MuleSoft learning path

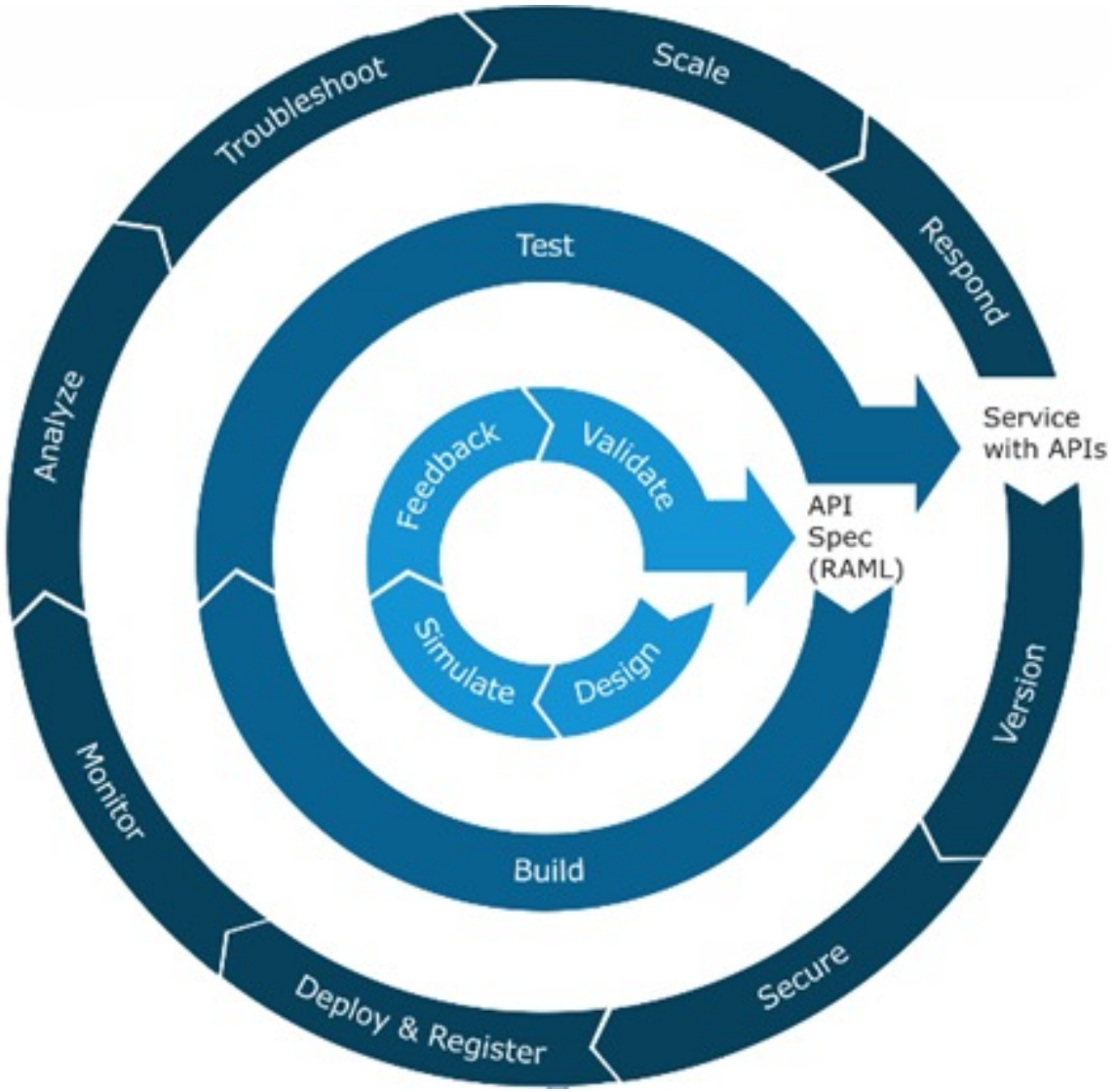


- Anypoint platform
  - Access management to deal with platform administration area's
  - Design API's using RAML or OAS
  - Dealing artifacts over exchange.
  - Cloud runtime to deploy mule applications.
  - API management. deploy mule apps.
  - Analytics management on deployed applications.
- Mule ESB integrations development using anypoint studio
- Dataweave language for transformations & enrichments
- MUnit to develop test cases driven development
- DevOps with mule
- Hands on real time project



- Integration architectural concepts
- Platform architectural concepts

Mule API life cycle



Setting up your computer

- Java installation & environment variables
- Maven installation & environment variables
- Setup Anypoint studio
- Postman installation

Web services

- **Web service : A web service is a program which runs in server and whose results are sharable over network/integration to make data transfer between applications.**
- **The types of web services are of two types SOAP and REST.**
- **SOAP services are legacy approach, and the development of SOAP services is very less comparing to REST.**
- **REST services are latest and called as REST API’s.**
- **The percentage of development of REST is 95% and SOAP is 5%**

How rest service can be accessed

- Before implementation of REST services, we should understand about a URL that we usually interact every day because we interact with REST API using URL.
    - http-protocol://host:port/base-path/sub-path
    - Example: http://localhost:8081/customer-sapi/customers
    - Example: http://mule-dev-customer-sapi.cloudhub.io/customer-sapi/customers
    - Example: https://mule-dev-customer-sapi.cloudhub.io/customer-sapi/customers

- HTTP Method : GET, POST, PUT, DELETE etc.
    - Example: GET https://mule-dev-customer-sapi.cloudhub.io/customer-sapi/customers
    - Example: POST https://mule-dev-customer-sapi.cloudhub.io/customer-sapi/customers
    - Example: PUT https://mule-dev-customer-sapi.cloudhub.io/customer-sapi/customers
    - Example: DELETE https://mule-dev-customer-sapi.cloudhub.io/customer-sapi/customers

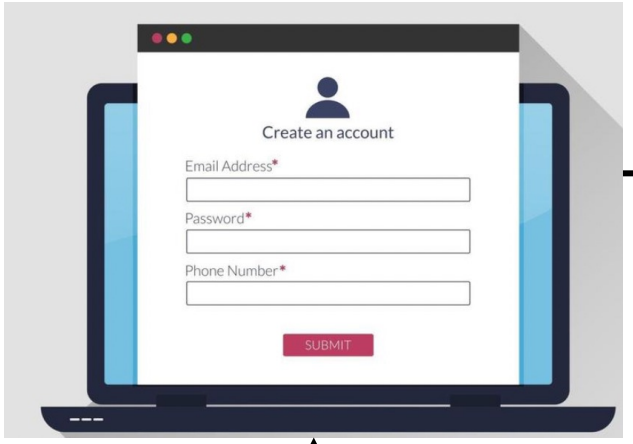
How request & response data passed over REST request

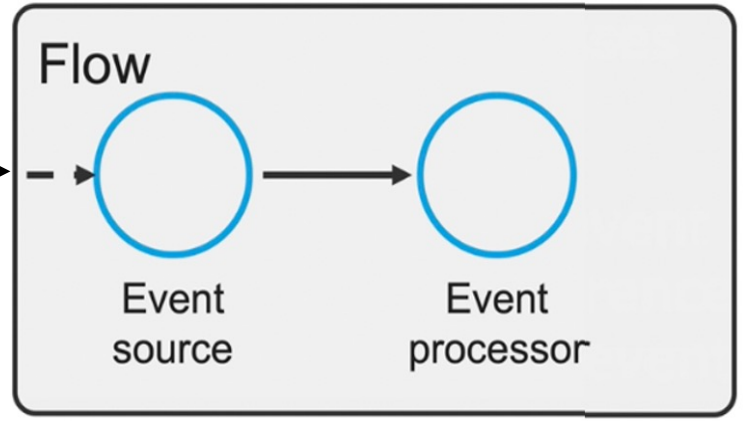
request headers

- Headers
- URI params
- Query params
- Etc.

request body

- JSON payload
- XML payload





Example: 200, OK. 400, Bad Request. 500 Server Error. etc.

response headers

- http response status code, status message

response body

- JSON payload
- XML payload

Process request and sends back response

Use case development of session : hello-mule-api

Rest configuration	Example
HTTP method	GET
Protocol	HTTP
Host	localhost
Port	8081
Base path	hello-mule-api
Sub path	hello-mule
URL	http://localhost:8081/hello-mule-api/hello-mule

Mule events

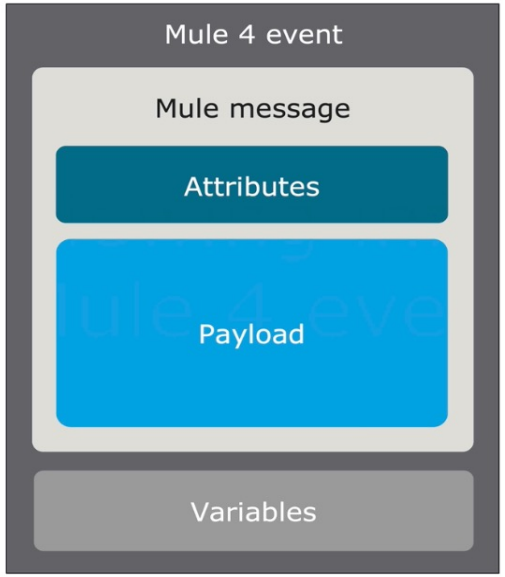
request headers

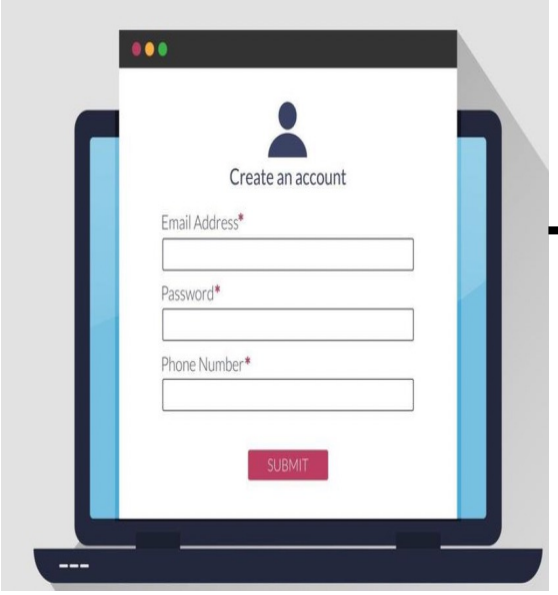
- Headers
- URI params
- Query params
- Etc.

request body

- JSON payload
- XML payload

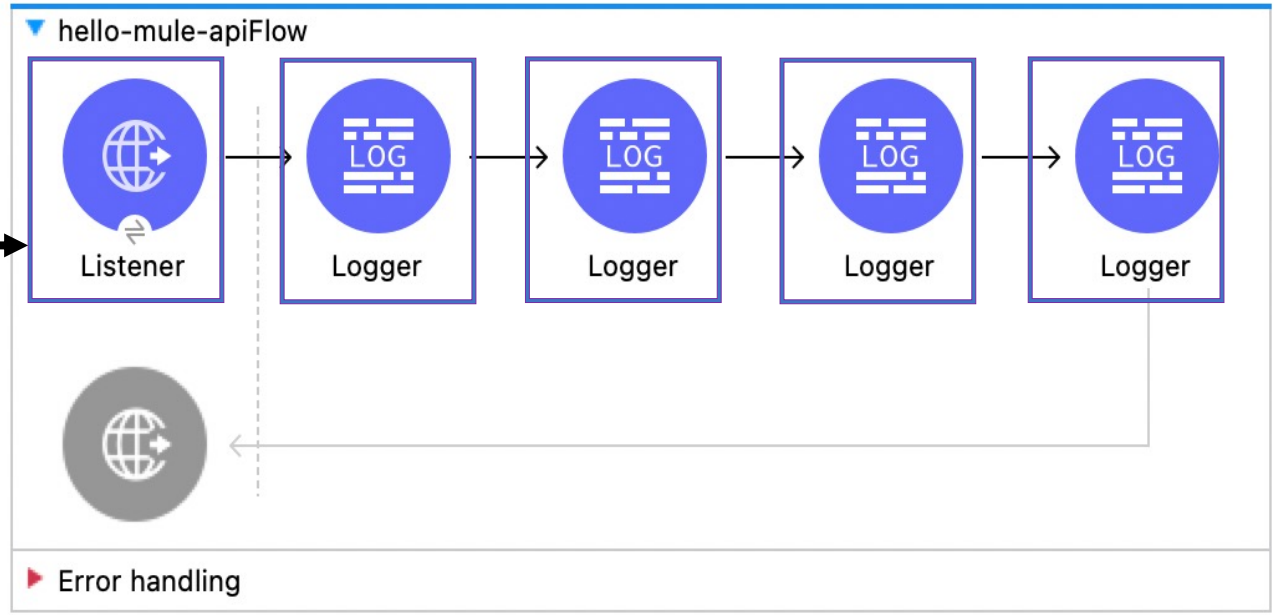
Mule event message





Mule event

Mule event source **Mule event processors**



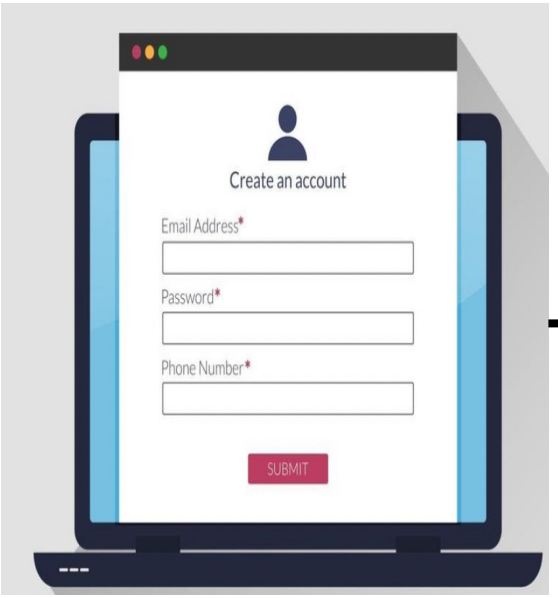
Mule message structure

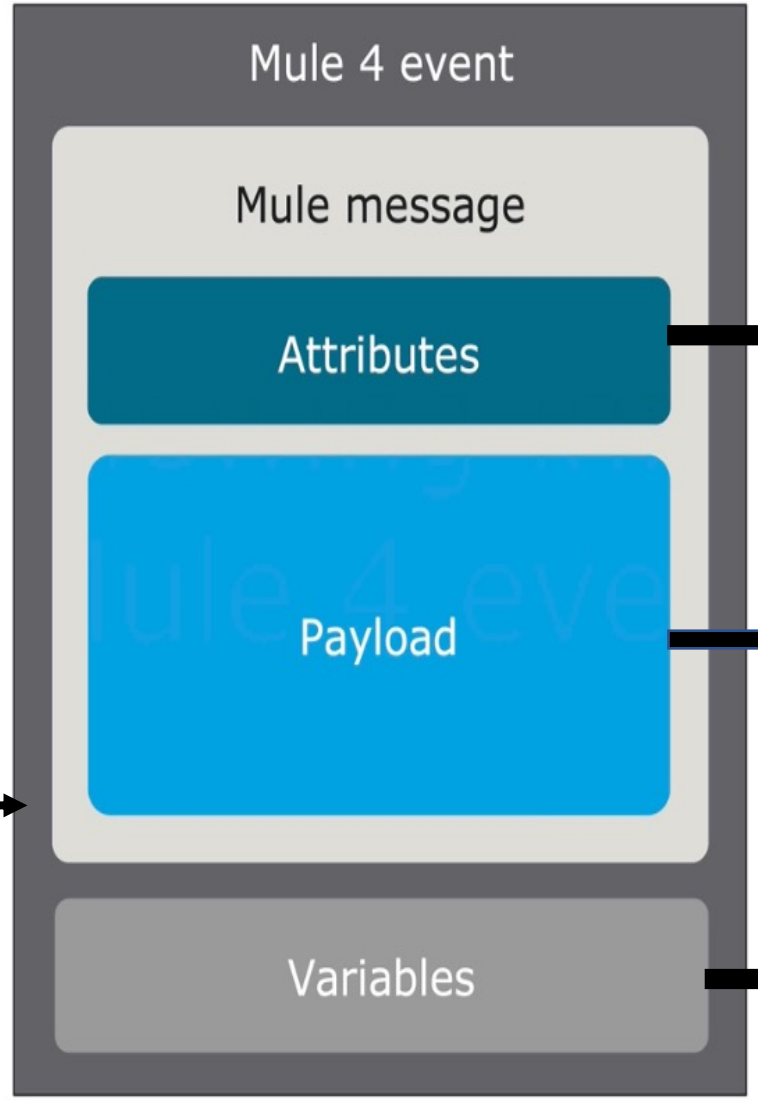
request headers

- Headers
- URI params
- Query params
- Etc.

request body

- JSON payload
- XML payload





attributes.headers.'Content-Type'

attributes.uriParams.fieldName

attributes.queryParams.fieldName

payload.fieldName

vars.variablename

create-employee rest service

- **URL:** http://localhost:8091/emp-sapi/create-employee (POST)

HTTP request body:

```
{
  "employeeID": 100,
  "employeeName": "Chinna",
  "employeeStatus": "A"
}
```

HTTP response header: 200, success

HTTP response body:

```
{
  "status": 200,
  "message": “Success”
}
```

update-employee rest service

- **URL:** http://localhost:8091/emp-sapi/update-employee (PUT)

HTTP request body:

```
<?xml version="1.0" encoding="UTF-8"?>
<employee>
        <employeeID>333</employeeID>
        <employeeStatus>A</employeeStatus>
</employee>
```

HTTP response body: HTTP response header: 200, OK

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
        <status>200</status>
        <message>success</message>
</response>
```

get-employee rest service

- **URL:** http://localhost:8091/emp-sapi/get-employee?employeeID=100 (GET)

  HTTP response header: 200, OK

  HTTP response body:

```
{
  “employeeID”: 100,
  “employeeName”: “Chinna”,
  “employeeStatus”: “A”
}
```

get-employees rest service

- **URL:** http://localhost:8091/emp-sapi/get-employees (GET)

HTTP response header: 200, OK

HTTP response body:

```
[
    {
        “employeeID”: 100,
        “employeeName”: “Chinna”,
        “employeeStatus”: “A”
    },
    {
        “employeeID”: 101,
        “employeeName”: “John”,
        “employeeStatus”: “A”
    }
]
```

delete-employee rest service

- **URL:** http://localhost:8091/emp-sapi/delete-employee/101/John (DELETE)

HTTP response header: 200, OK

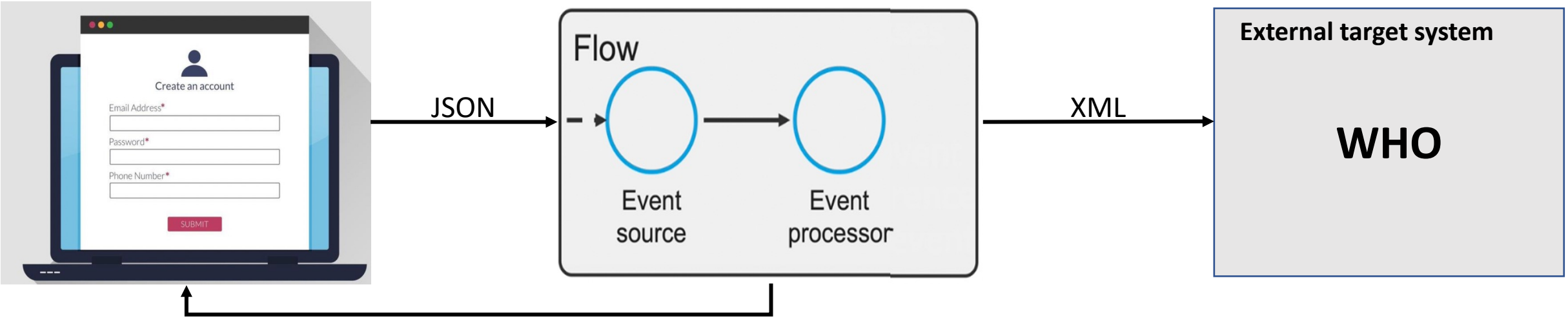
HTTP response body:

```
{
    “status”: 200,
    “message”: “Success”
 }
```

ESB standards

ESB integration tool: A integration tool called as ESB tool when it provides below features.

- Components to orchestrate integrations
- Transformations
- Enrichments



Transformations & Enrichments – Mule components & Transform message connector

Dataweave logic

JSON

```
{
	"source": "DPC Hospital",
	"caseType": "positive",
	"firstName": "John",
	"lastName": "Nixon",
	"phone": "541-754-3010",
	"email": "john@gmail.com",
	"dateOfBirth": "1989-04-26",
	"nationalID": "209-49-6193",
	"nationalIDType": "SSN",
	"address": {
		"streetAddress": "1600 Pennsylvania Avenue",
		"city": "Dallas",
		"state": "TX",
		"postal": "20500",
		"country": "USA"
	}
}
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<COVID_CASE>
  <FIRST_NAME>John</FIRST_NAME>
  <LAST_NAME>Nixon</LAST_NAME>
  <PHONE>541-754-3010</PHONE>
  <EMAIL>john@gmail.com</EMAIL>
  <DATE_OF_BIRTH>1989-04-26</DATE_OF_BIRTH>
  <COUNTRY>usa</COUNTRY>
</COVID_CASE>
```

Dataweave logic

JSON

```
{
	"source": "DPC Hospital",
	"caseType": "positive",
	"firstName": "John",
	"lastName": "Nixon",
	"phone": "541-754-3010",
	"email": "john@gmail.com",
	"dateOfBirth": "1989-04-26",
	"nationalID": "209-49-6193",
	"nationalIDType": "SSN",
	"address": {
		"streetAddress": "1600 Pennsylvania Avenue",
		"city": "Dallas",
		"state": "TX",
		"postal": "20500",
		"country": "USA"
	}
}
```

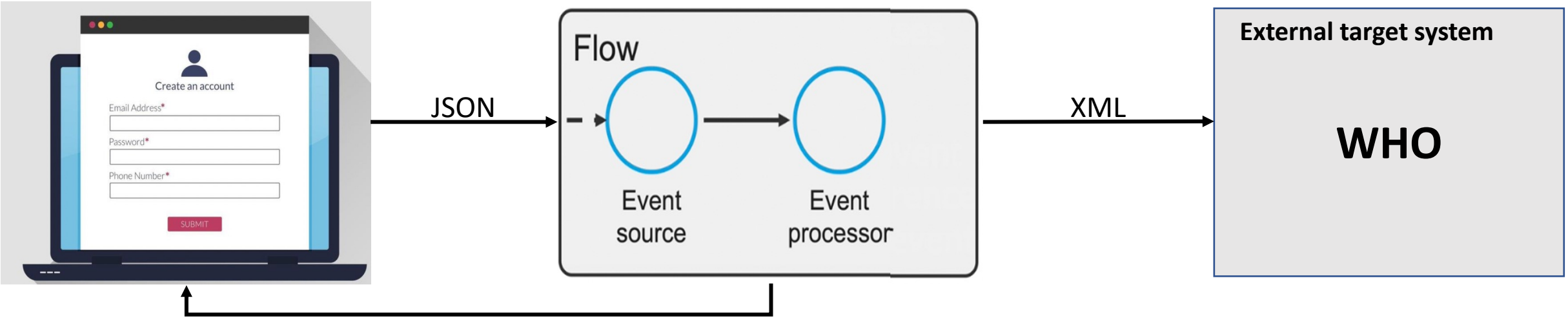
DW

```
%dw 2.0
output application/xml
---
{
COVID_CASE: {
    FIRST_NAME: payload.firstName,
    LAST_NAME: payload.lastName,
    PHONE: payload.phone,
    EMAIL: payload.email,
    DATE_OF_BIRTH:
    payload.dateOfBirth,
    COUNTRY: payload.address.country
  }
}
```

DataWeave Introduction

ESB integration tool: A integration tool called as ESB tool when it provides below features.

- Components to orchestrate integrations
- Transformations
- Enrichments



Transformations & Enrichments – Mule components & Transform message connector using **DataWeave** language

DataWeave selectors

DataWeave selectors: To extract the fields from mule message structure.

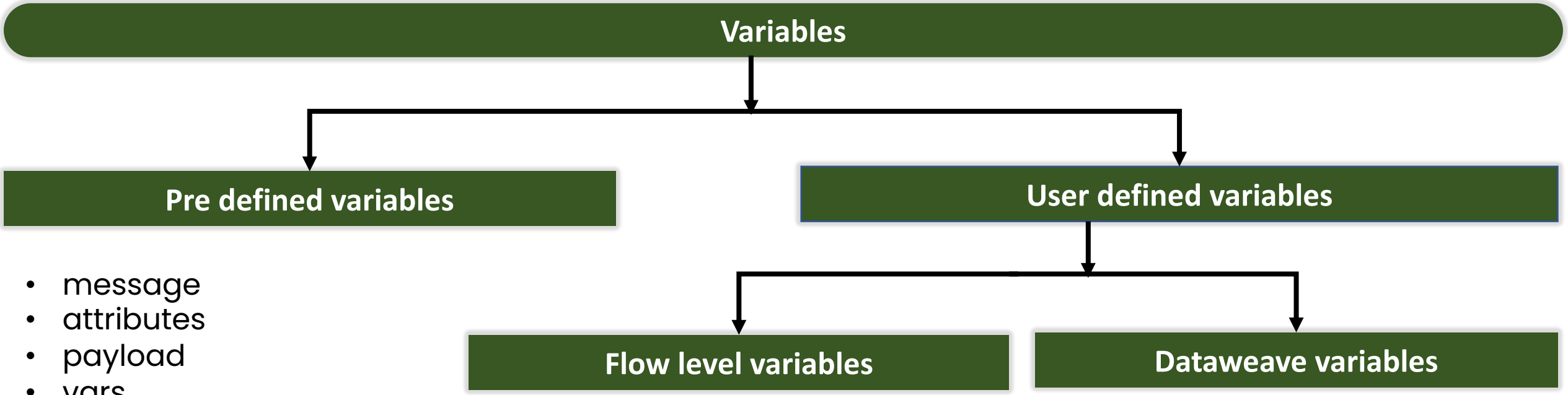
Selector	Syntax
Single-value	.keyName
Multi-value	.*keyName
Index	[<index>]
XML attribute	.@keyName
Range	[<index> to <index>]
Filter	[?(boolean_expression)]
Namespace	keyName.#
Descendants	..keyName
Dynamic	payload[(nameExpression)]
Key-value pair	.&keyName
Key present	keyName?, keyName.@type?
Assert present	keyName!
Metadata	.^someMetadata

Mule data types

Data types: Mule supports below datatypes.

Data type	Example
String	"hello world"
Number	123
VeryBigNumber	123412341341234123412341234123
FloatingPointNumber	123.456
Boolean	**true**
Date	2018-12-07
Time	11:55:56
DateTime	2018-10-01T23:57:59-03:00
LocalDateTime	2018-10-01T23:57:59-03:00
Array	[1, 2, 3, 5, 8]
MixedArray	[1, 2, "blah”]
Object	{ ”caseID": ”267" }

Types of variables



- message
- attributes
- payload
- vars
- error
- app
- flow
- mule
- server

- The scope of variables are application level

- The scope of variable is flow level

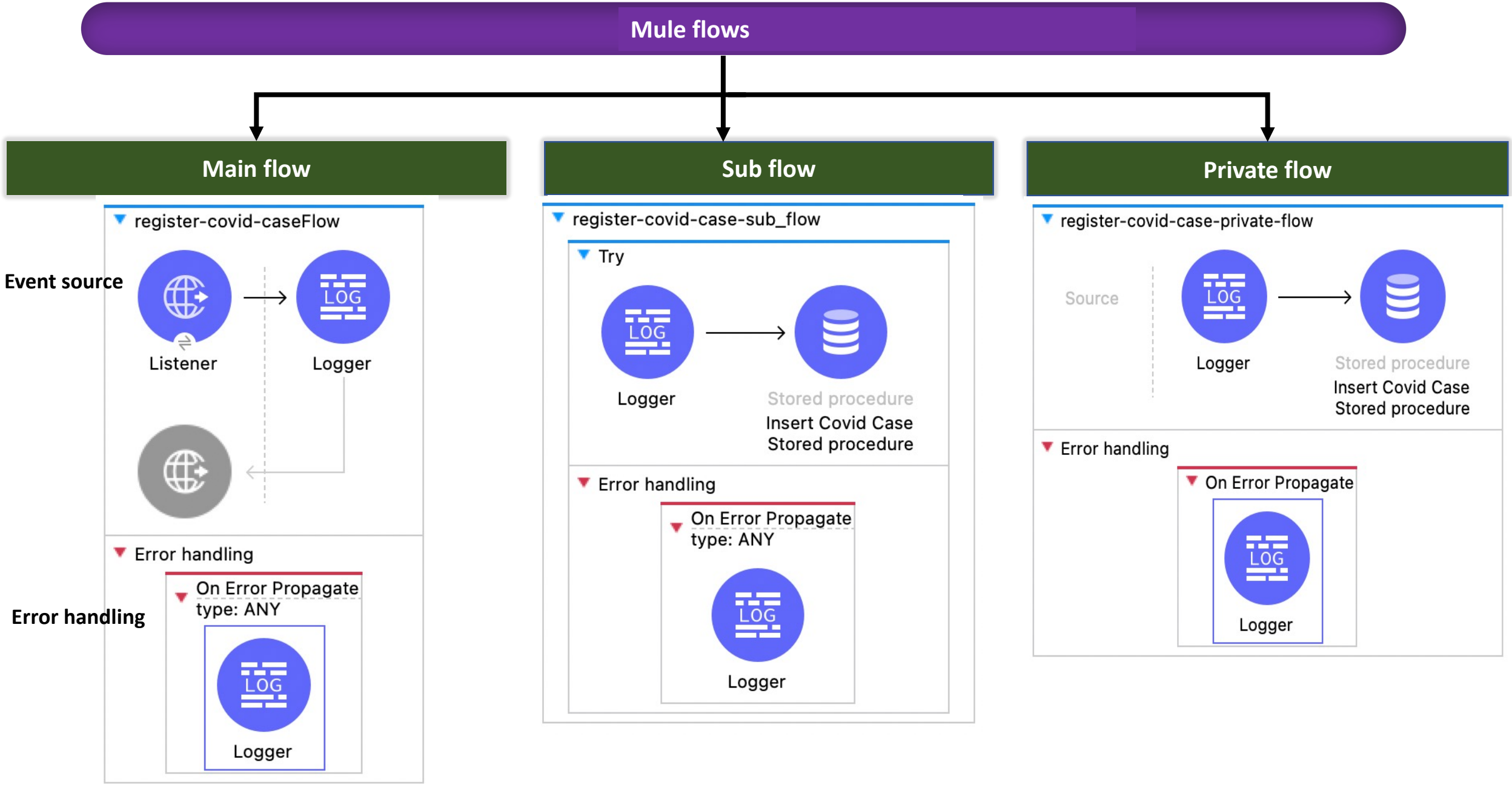
- The scope of variable is dw level

Type coercion

Type coercion: The process of changing field value from one type to other type

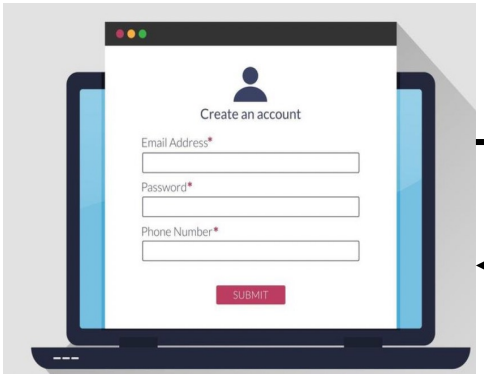
- Types can be coerced from one type to other using the ‘as’ operator
- Input : {“caseID”: “667”}
- Type coercion from string to number : payload.caseID as Number

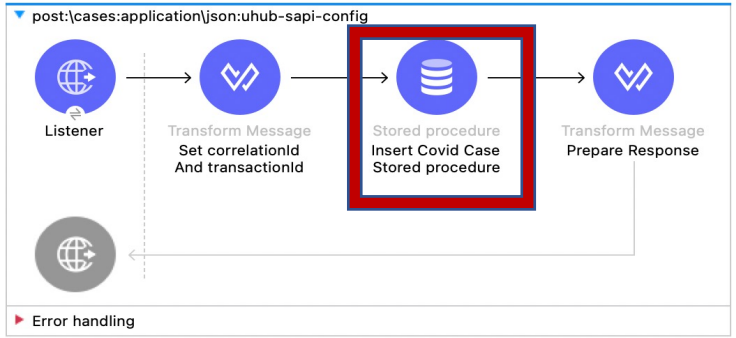
Types of mule flows



Error handling

- **Error handling: The process of handling errors in flow and responding back user friendly messages instead of system error messages is known as error handling.**





response headers

500, Server error

response body

Could not obtain connection from data source

response headers

503, Service unavailable

response body


```
{
  "code":503,
  "message":"Service unavailable",
  "description":"The service is temporarily not available",
  "dateTime":"2024-05-31T06:18:02Z",
  "transactionId":"48n32920-69ne-47b4-907d-fa15869f3c4d"
}
```

Error handling

Description

It is an important component of Mule error which will give description about the problem. Its expression is as follows:

```
#[error.description]
```

Type

The **`Type`** component of Mule error is used to characterize the problem. It also allows routing within an error handler. Its expression is as follows:

```
#[error.errorType]
```

Cause

The **`Cause`** component of Mule error gives the underlying java throwable that causes the failure. Its expression is as follows:

```
#[error.cause]
```

Message

The **`Message`** component of Mule error shows an optional message regarding the error. Its expression is as follows:

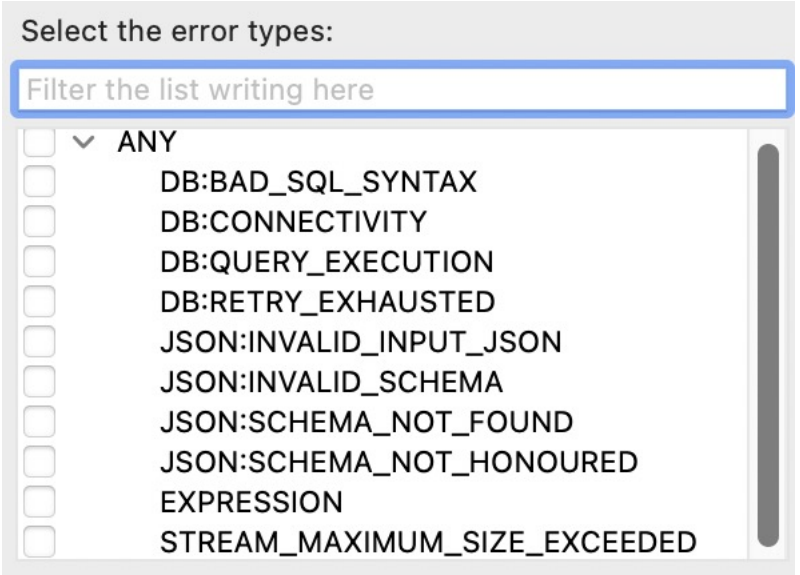
```
#[error.errorMessage]
```

Child Errors

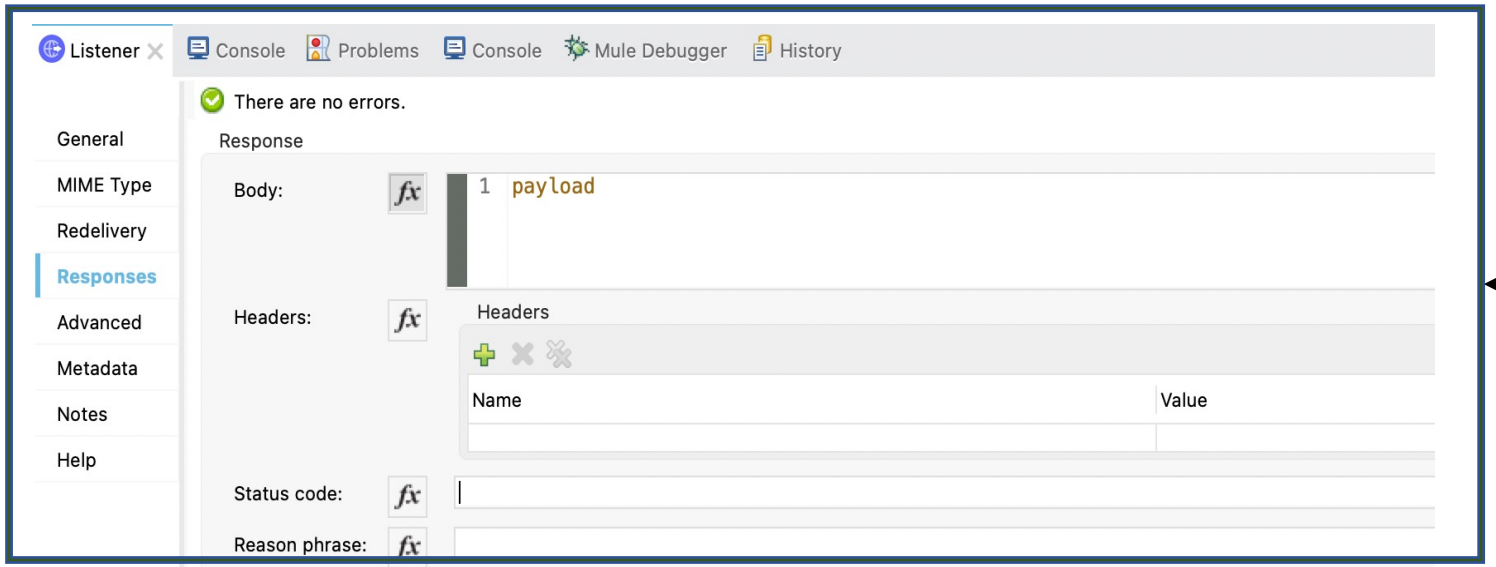
The **`Child Errors`** component of Mule error gives an optional collection of inner errors. These inner errors are mainly used by elements like Scatter-Gather to provide aggregated route errors. Its expression is as follows:

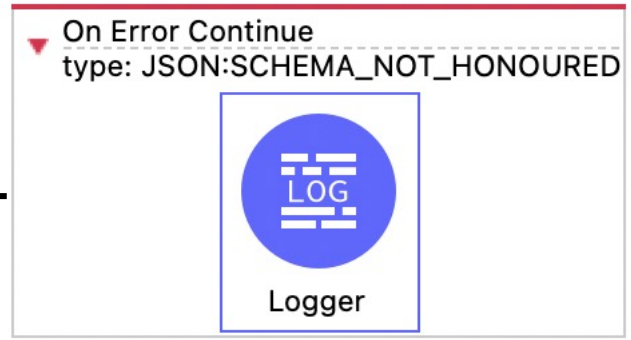
```
#[error.childErrors]
```

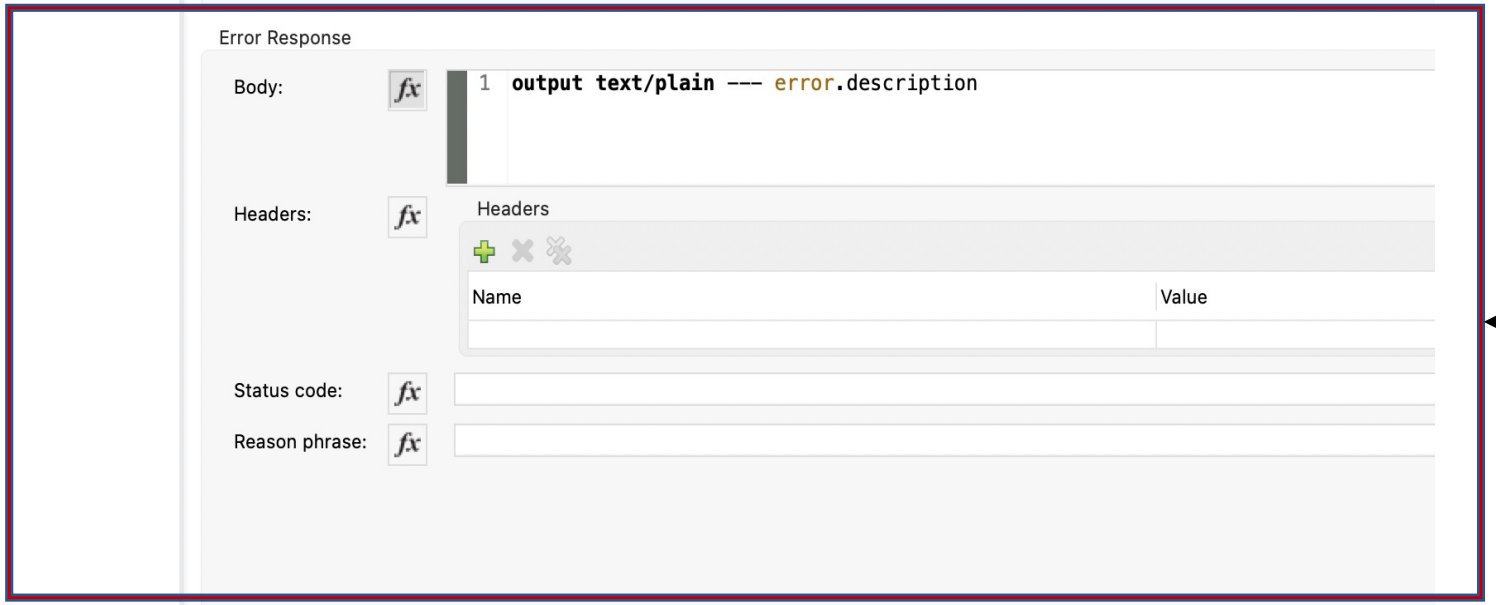
NAMESPACE:IDENTIFIER

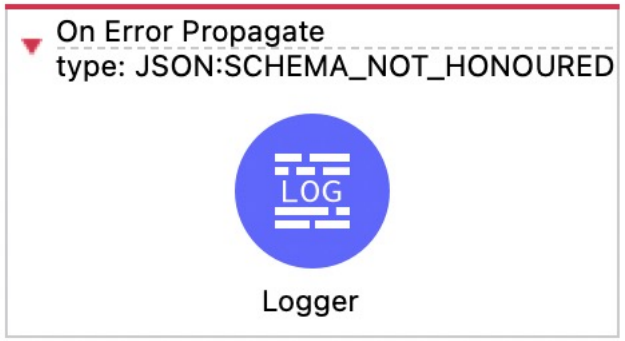


On error propagate vs on error continue



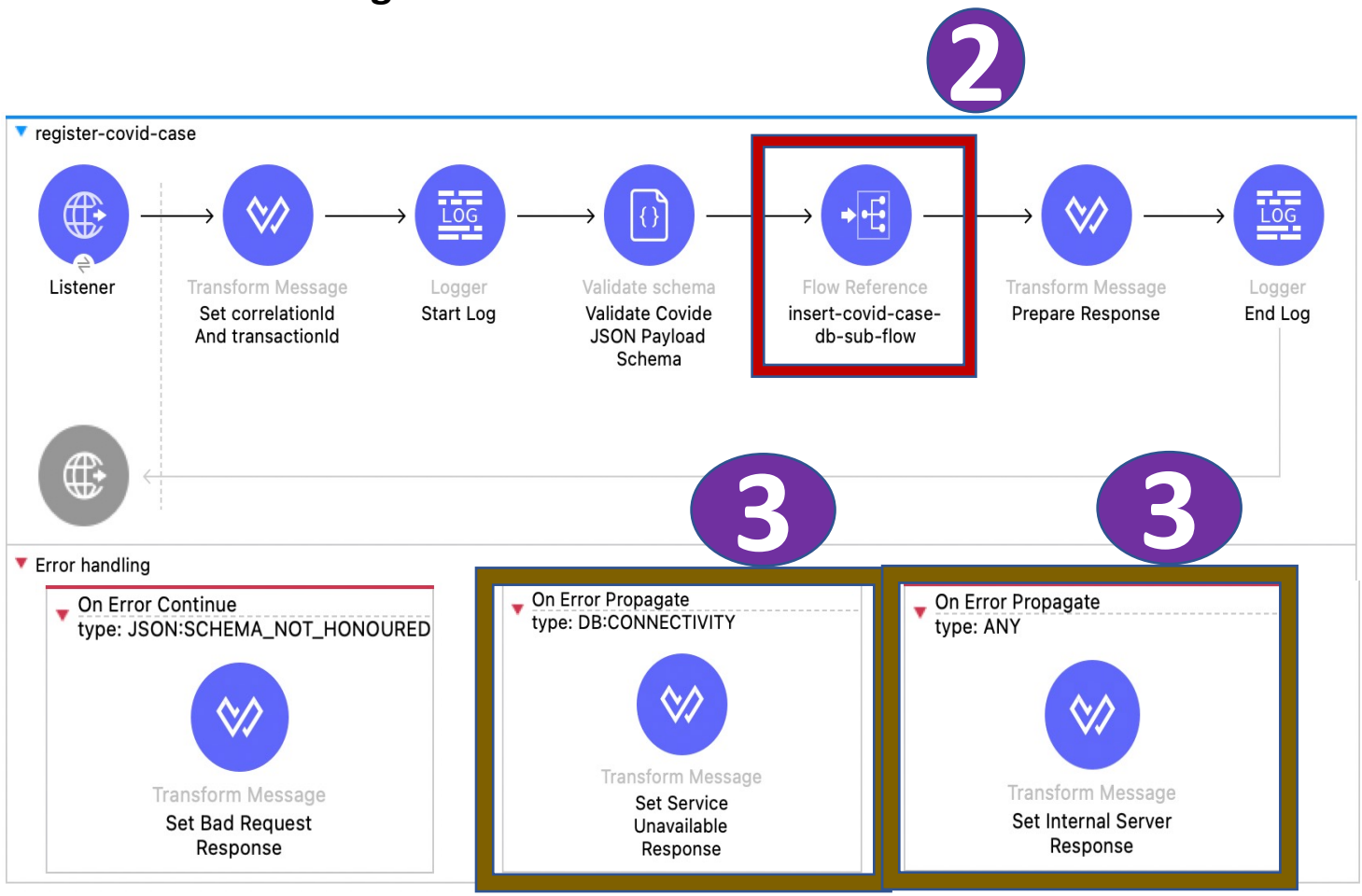


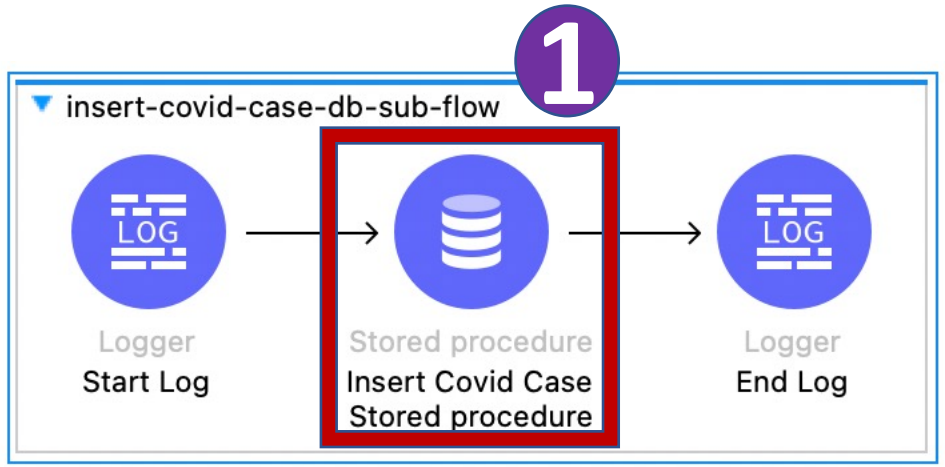




Error handling b/w parent flow & sub flow

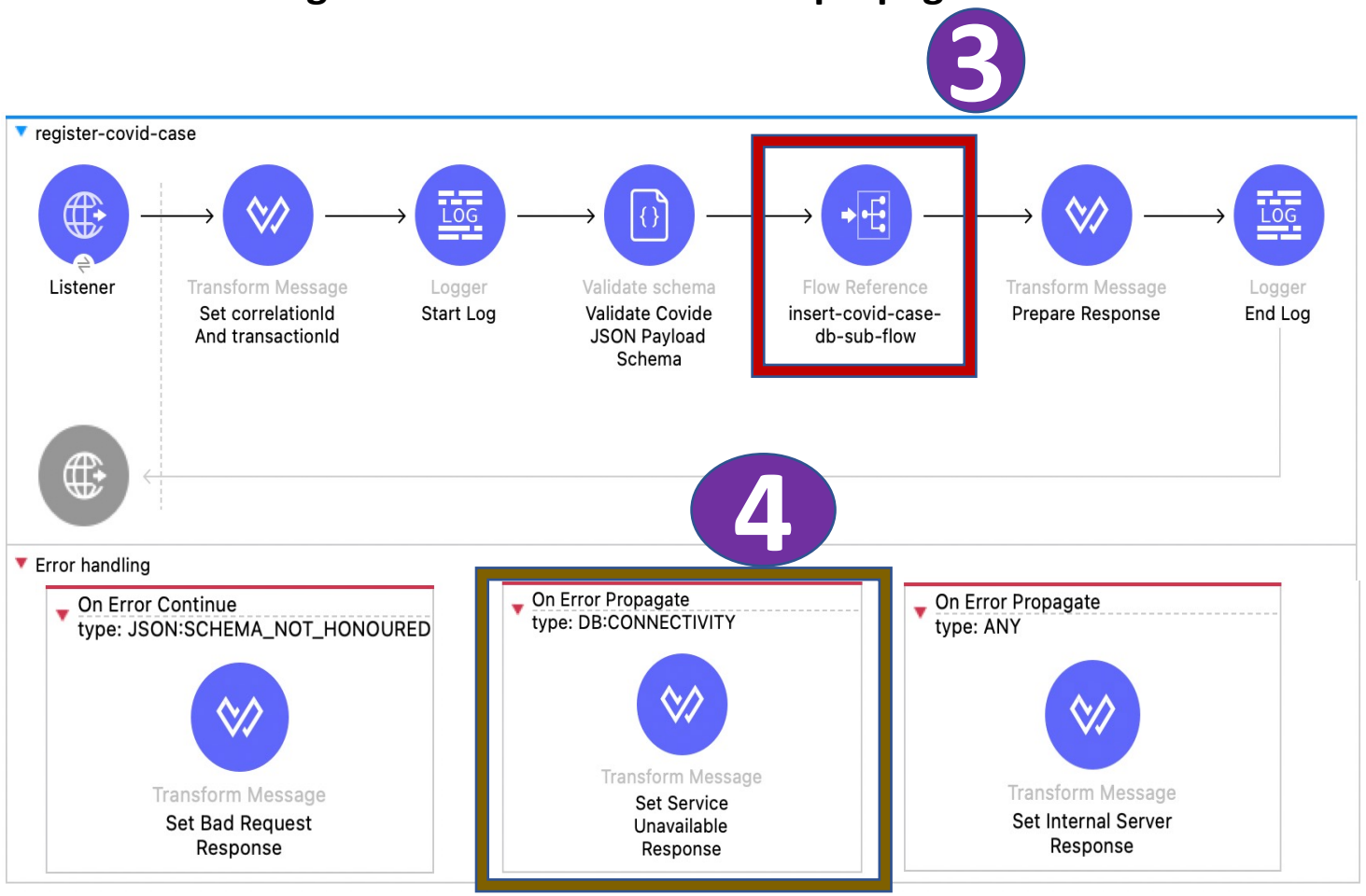
1. No error handling sub flow

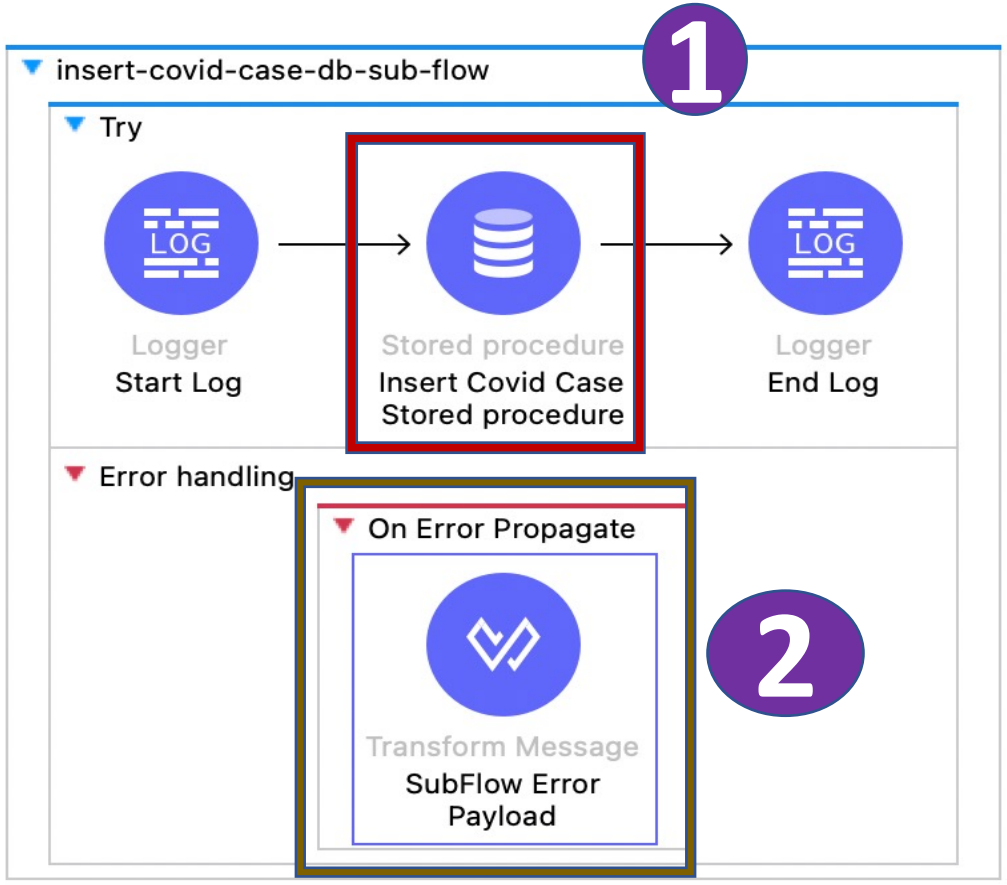




Error handling b/w parent flow & sub flow

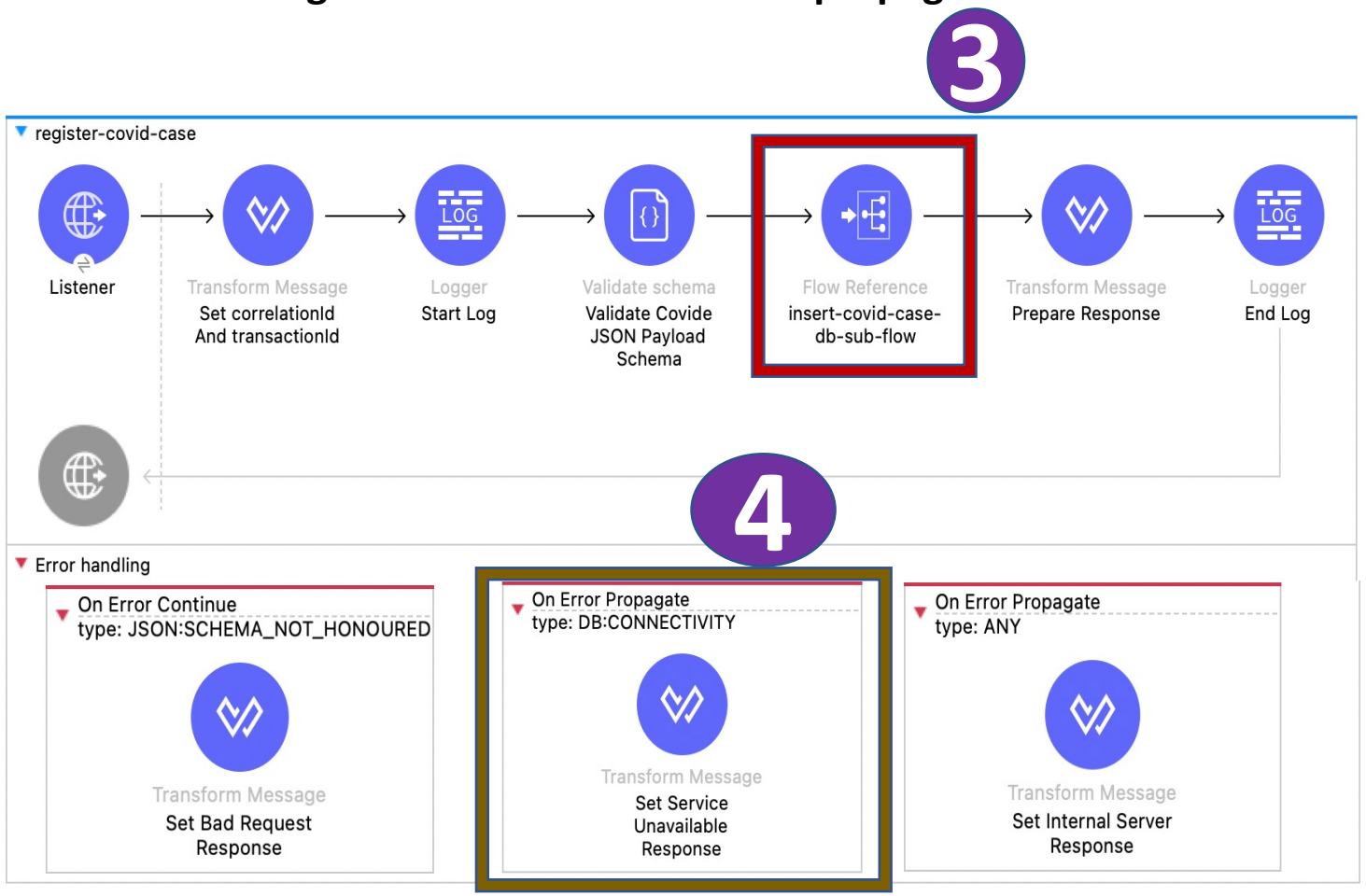
2. Error handling in sub flow with on-error-propagate

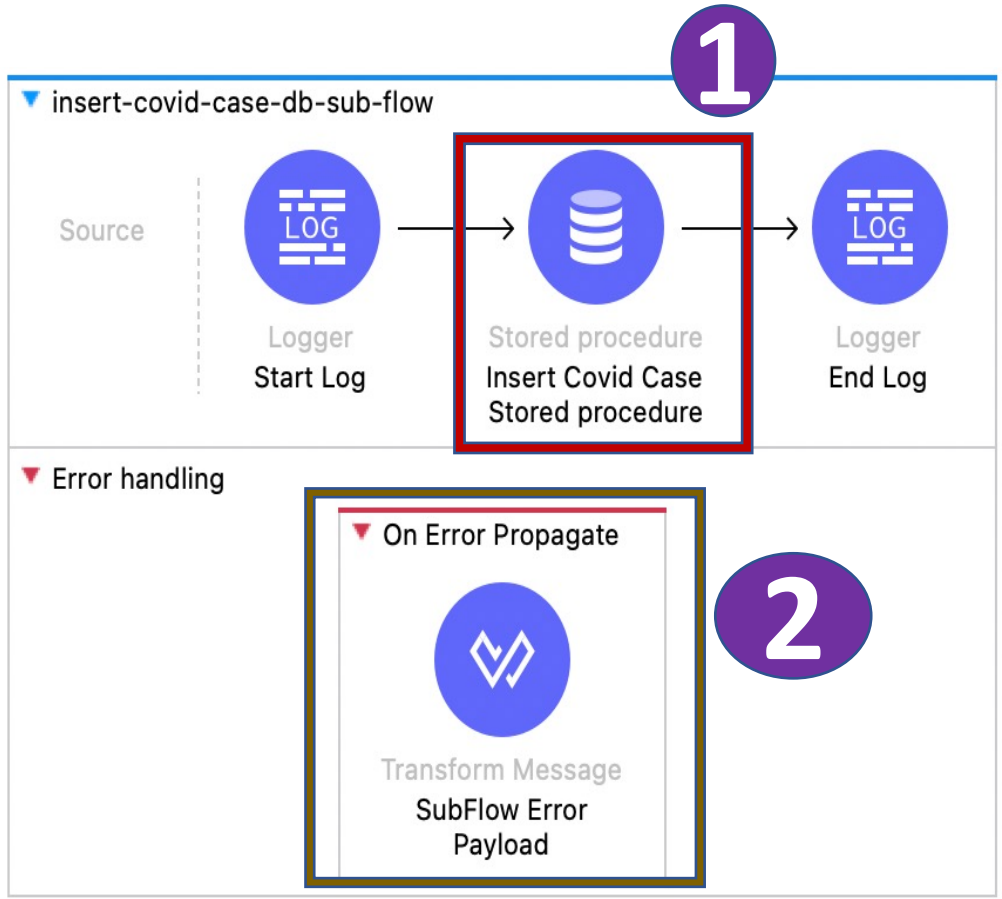




Error handling b/w parent flow & sub flow

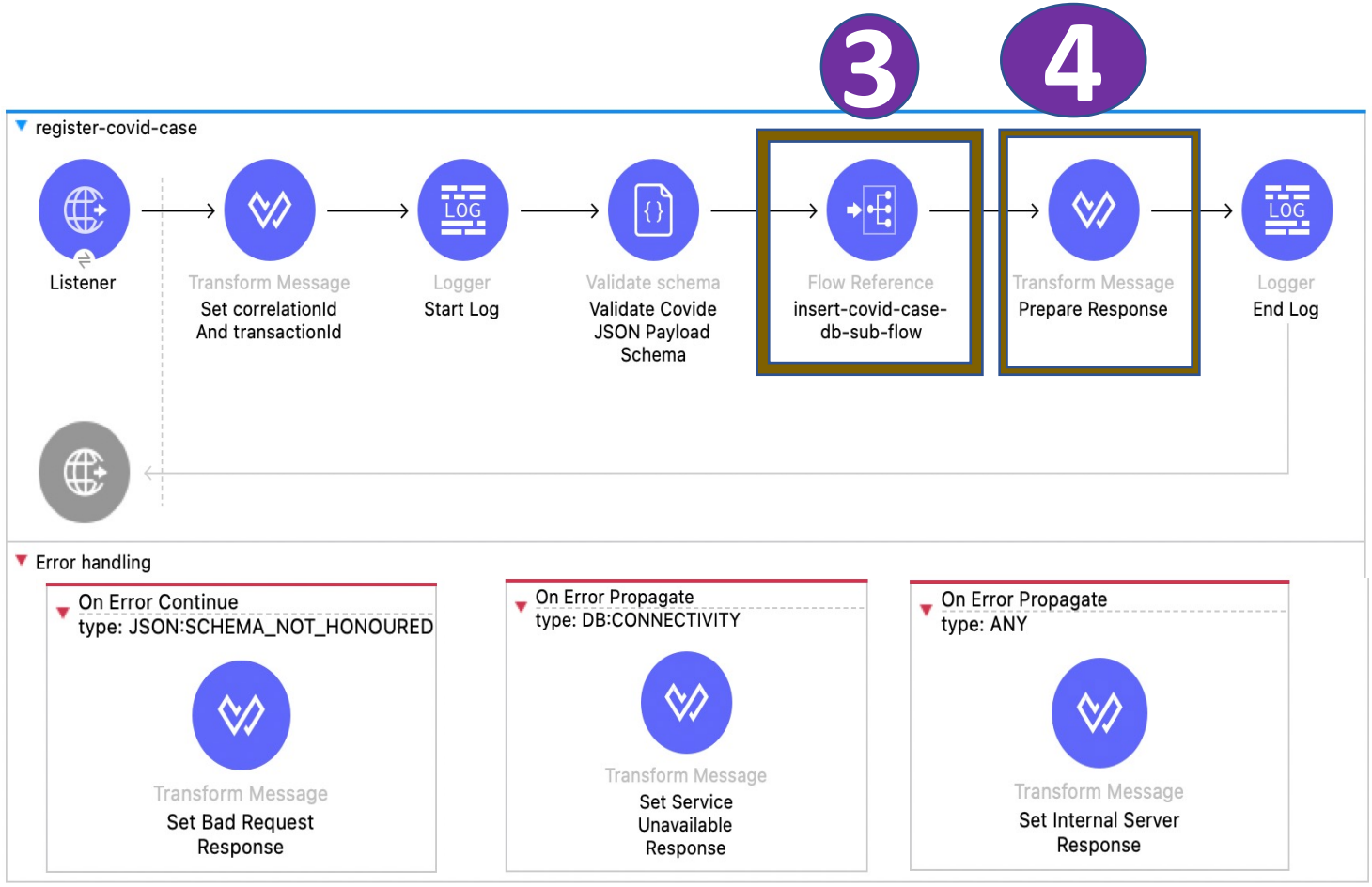
2. Error handling in sub flow with on-error-propagate

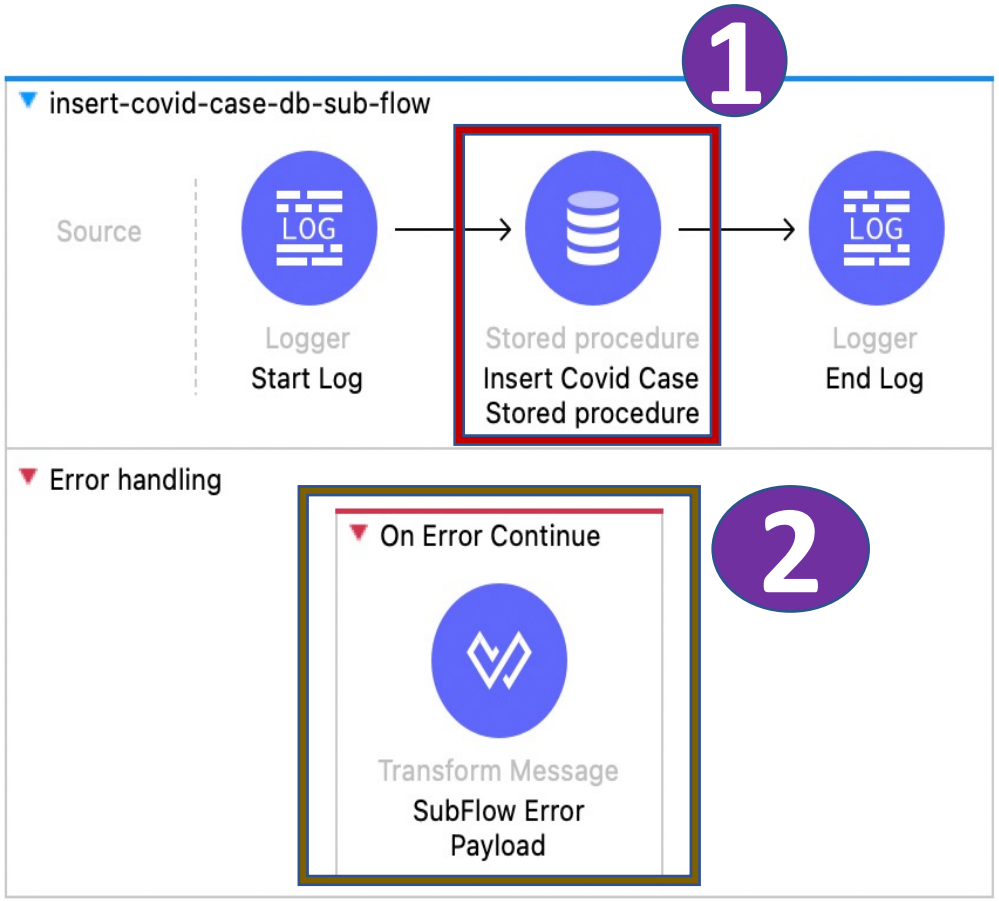




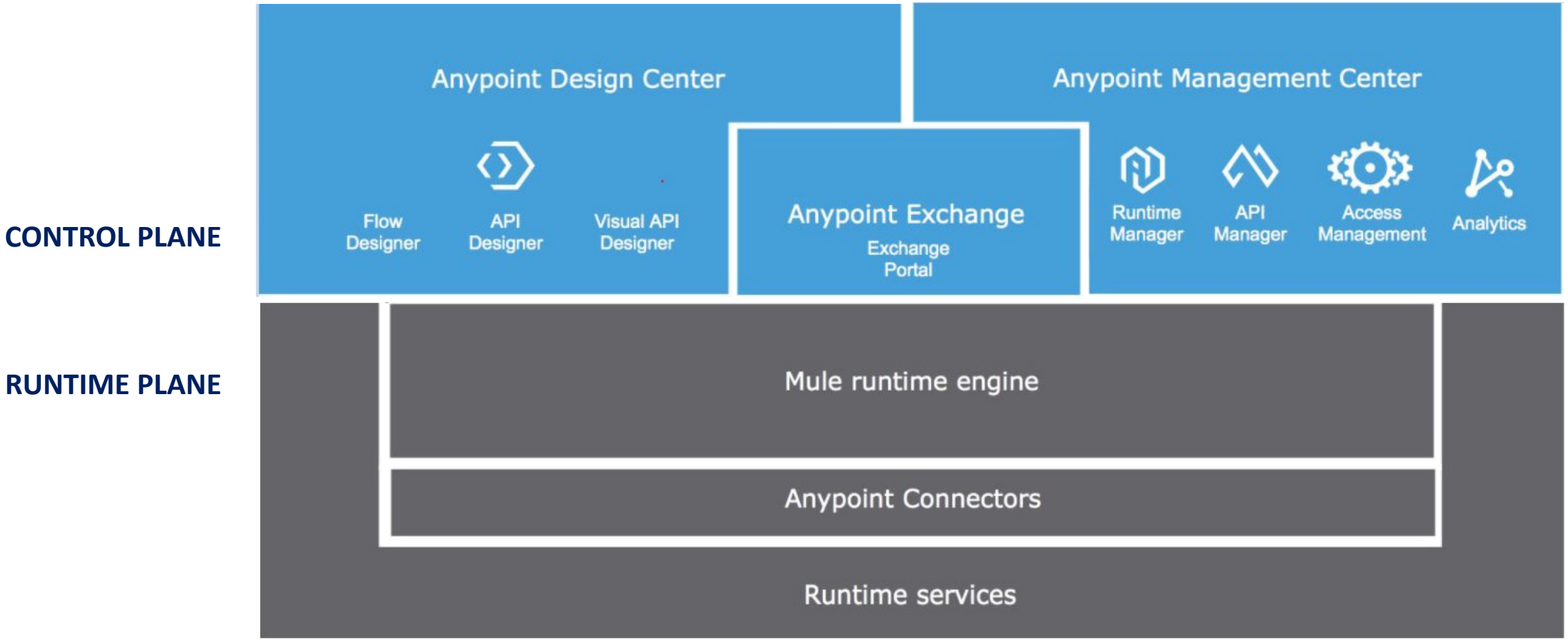
Error handling b/w parent flow & sub flow

3. Error handling in sub flow with on-error-continue

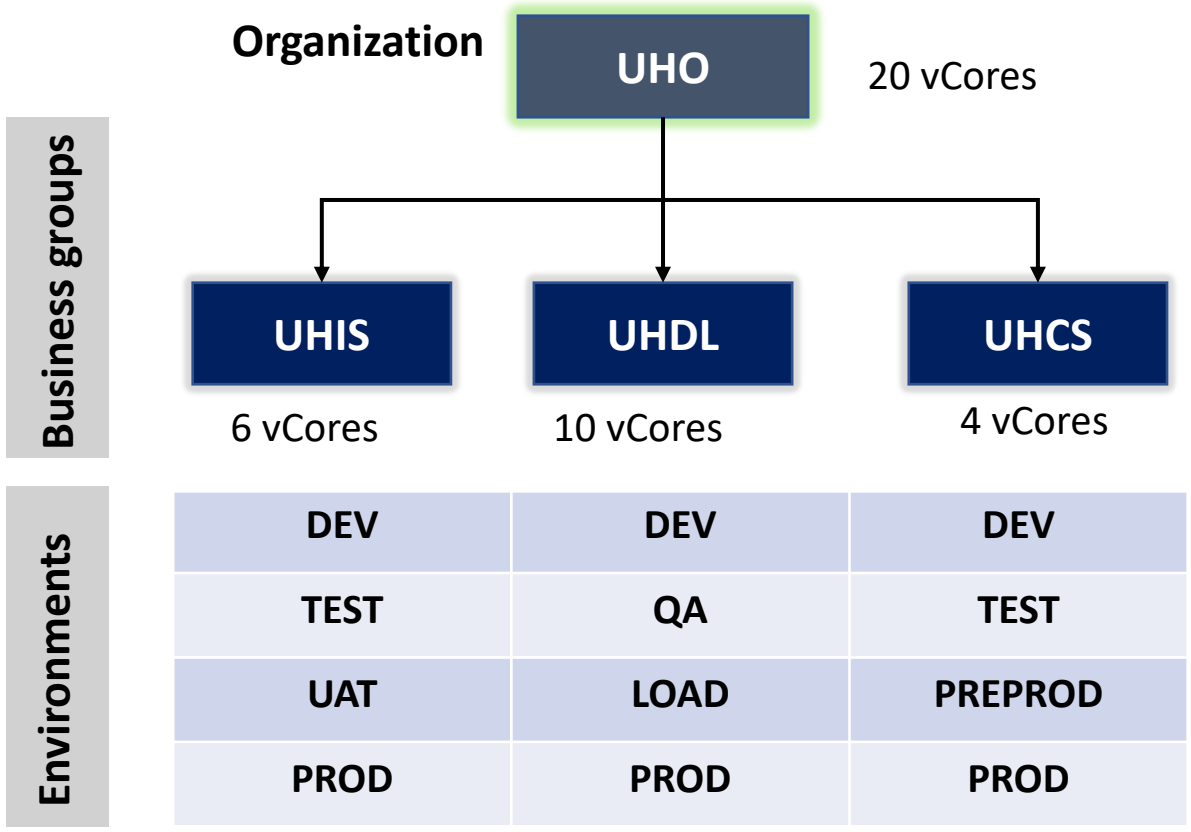




Anypoint platform capabilities

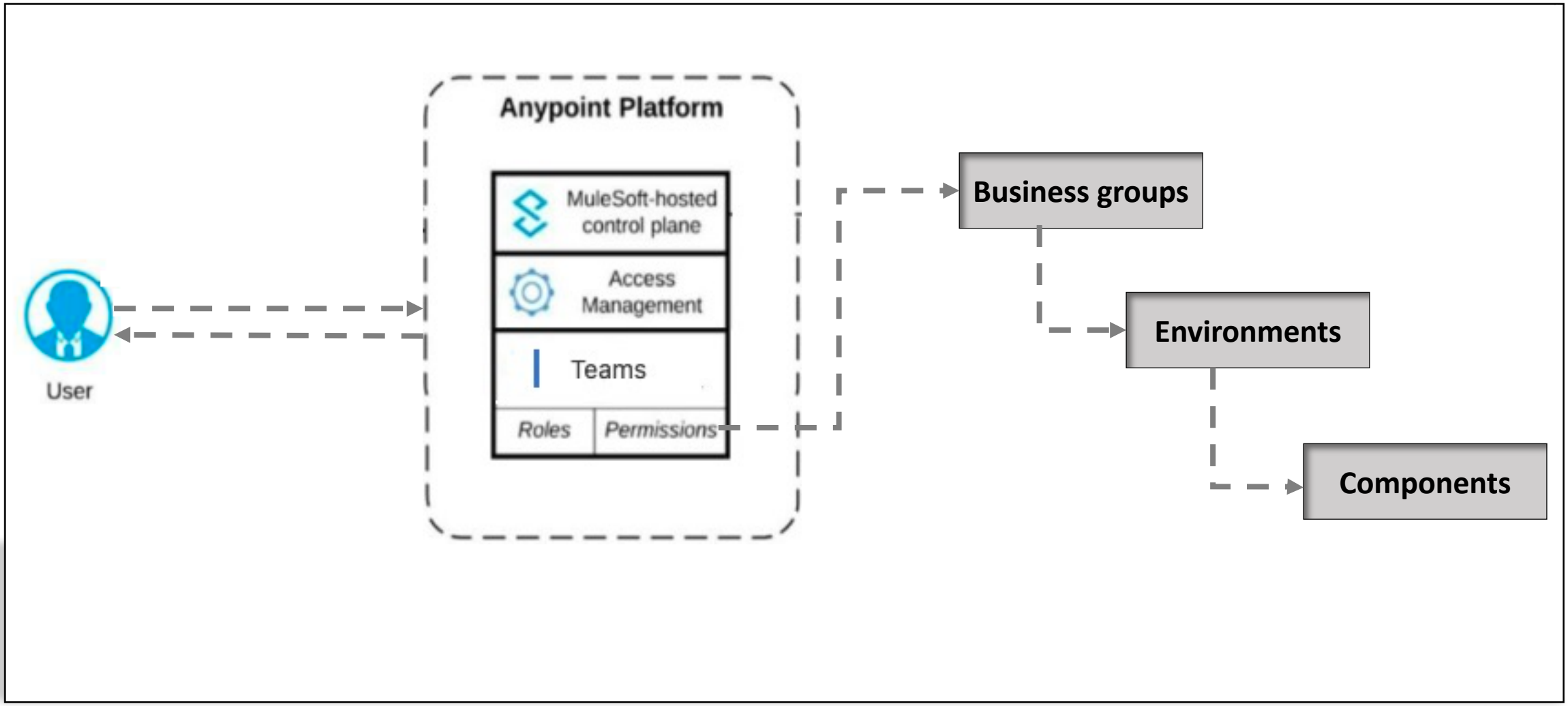


Organization, Business groups and Environments

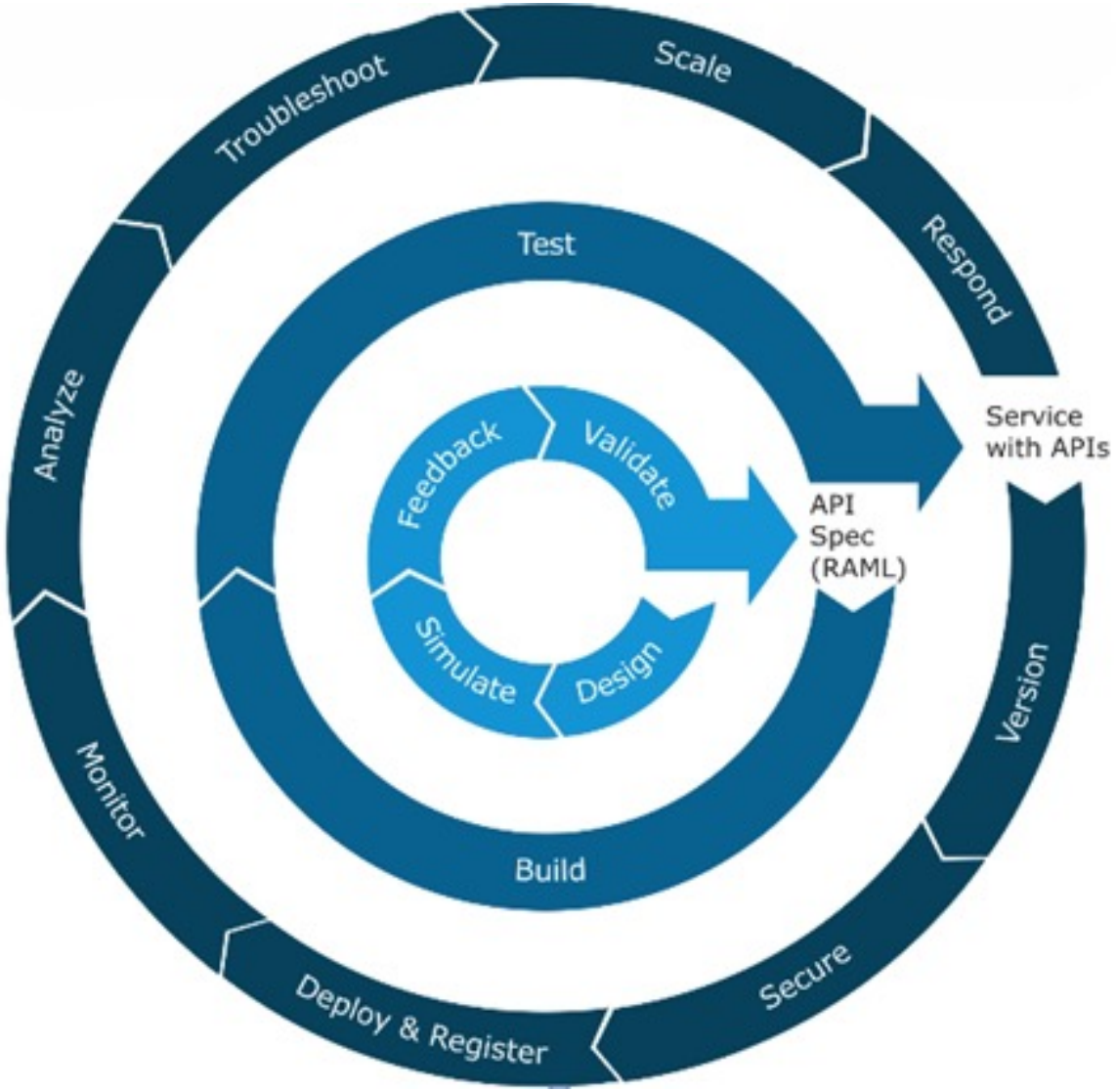


- The owner of a parent business group automatically has and retains administrator permissions for any child business group of that parent.
- Owners of child business groups cannot access or modify the parent business group or master organization settings.
- The APIs are individual to each business group, they can't be accessed one from other.
- The configuration of business group consist business group name, owner, owner can create child business groups, owner can create environments in it, vCores for production and vCores for Sandbox.

Anypoint platform user management



Mule API life cycle



DESIGN

↓

DEVELOPMENT

↓

MANUAL UNIT TESTING

↓

MUNIT TESTING

↓

MANUAL DEPLOYMENT

↓

CICD DEPLOYMENT

↓

API MANAGEMENT

BIG UPDATE ON MULE RUNTIME FOR COMMUNITY EDITION USERS

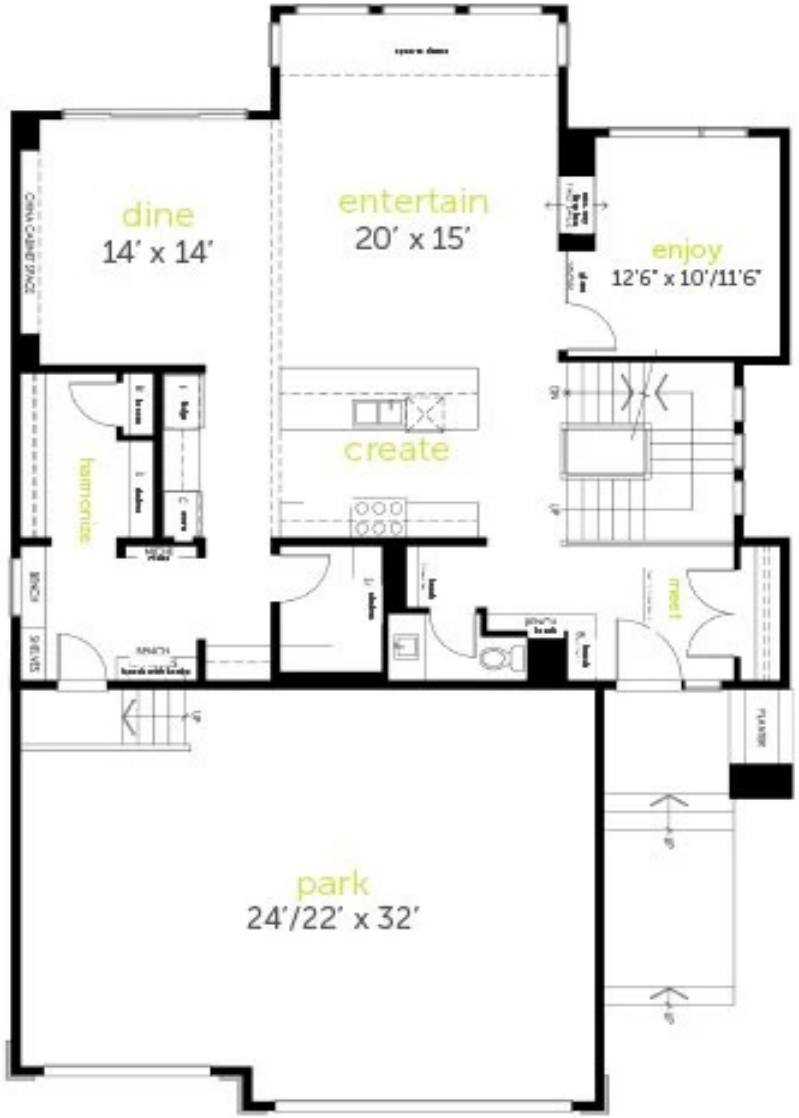
MULESOFT supports both CloudHub 1.0 & CloudHub 2.0

Runtime Version	Community Edition	Enterprise Edition
CloudHub 1.0	✗	✓
CloudHub 2.0	✓	✓

Designing API's

Designing API's

Mecca Main floor: 1407 sq.ft



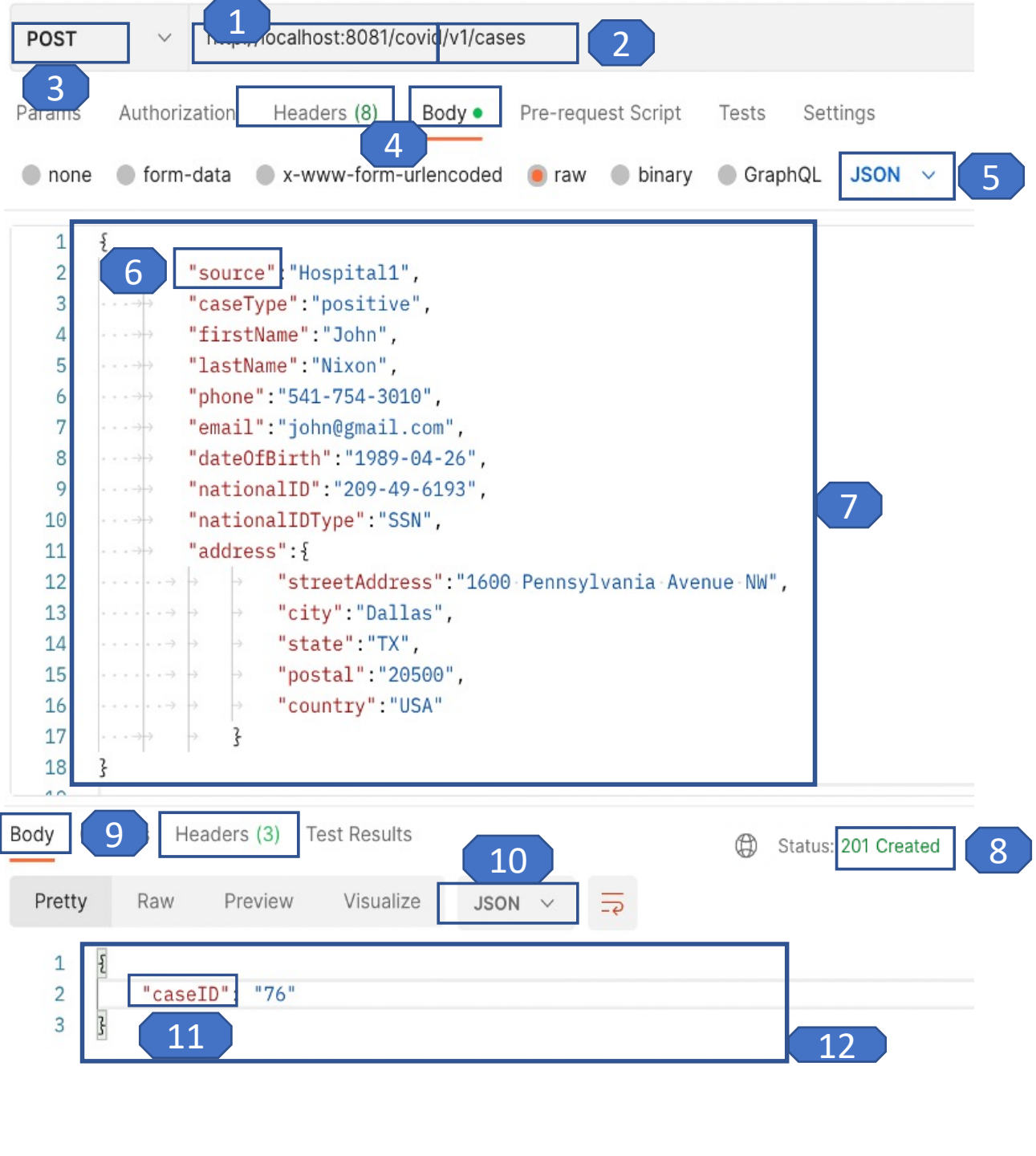
Aurora Main floor: 1445 sq.ft

Rest API

Base URI

Resource

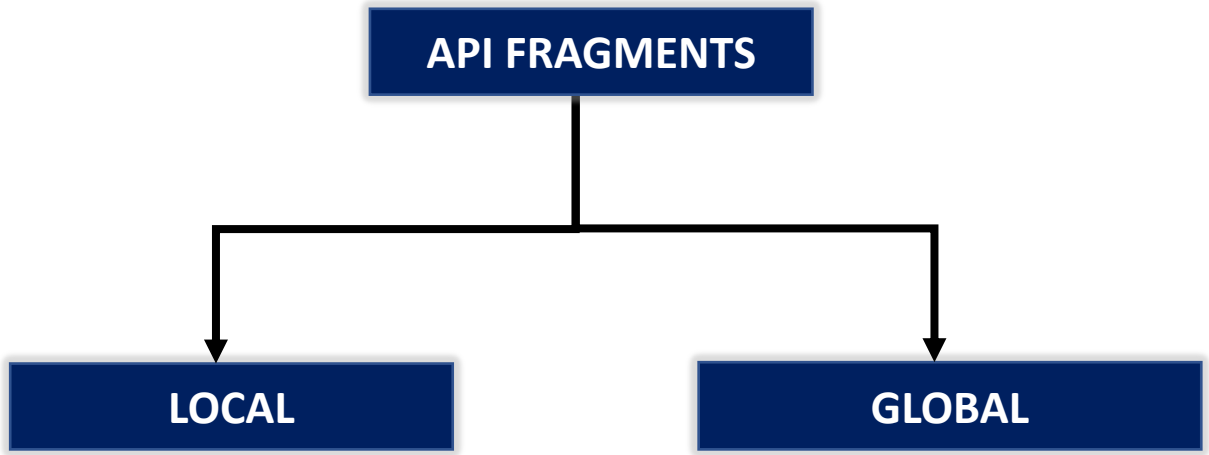
- REST API URL : http://localhost:8081/covid/v1/cases
- HTTP Method : GET, POST, PUT, DELETE etc.
- Input: headers, body
- Content type: application/json, application/xml etc.
- Response headers: 200, OK. 400, Bad Request. 500 Server Error. Etc
- Response body: application/json, application/xml etc.




```yaml
#%RAML 1.0
title: uhub-sapi
baseUri: http://localhost:8081/covid
/v1/cases:
  post:
    body:
      application/json:
        properties:
          source:
            type: string
            required: true
            example: "Hospital1"
          .
        example:
          {
            "source":"Hospital1",
            .
            .
          }
    responses:
      200:
        body:
          application/json:
            properties:
              caseID:
                type: string
                example: "76"
            example:
              {
                "caseID": "76"
              }
```

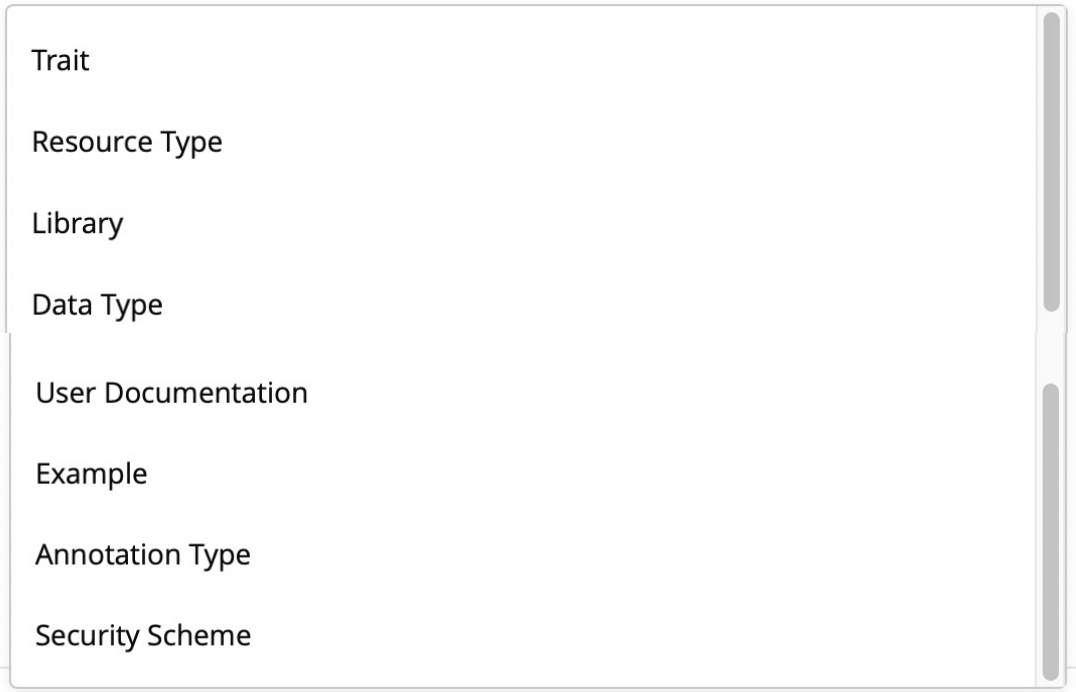
Fragments in RAML

Fragments : API fragments are reusable component of RAML to make the design and build of a reusable API even quicker and easier.

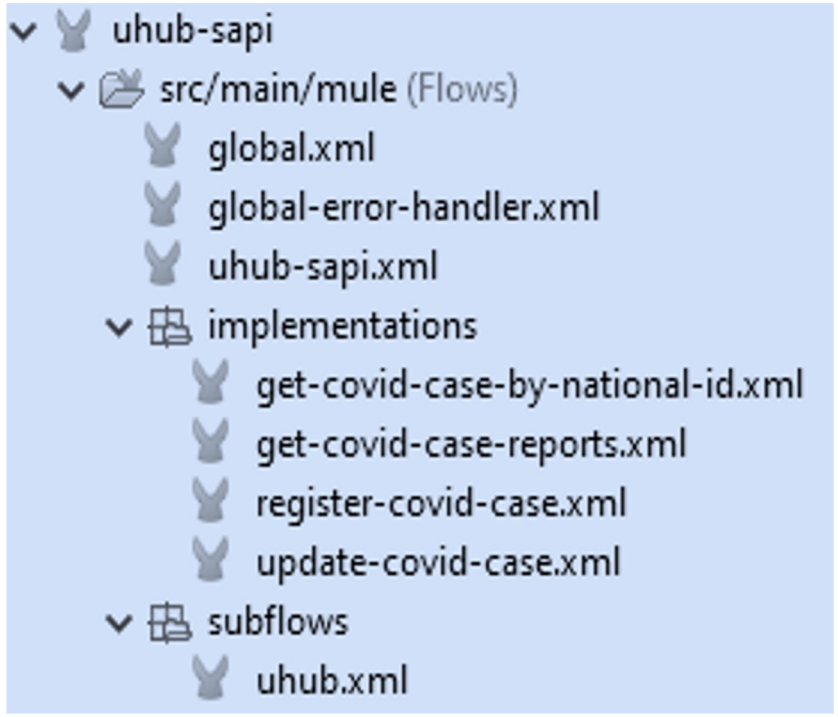


uhub-sapi

uhub-sapi
aws-sapi
who-sapi



Mule project naming standards and structure



Project name: kebab case (ex: uhub-sapi)

Flow and subflow names: kebab case (ex: register-covid-case)

Property file names: kebab case (ex: uhub-sapi-dev.yaml)

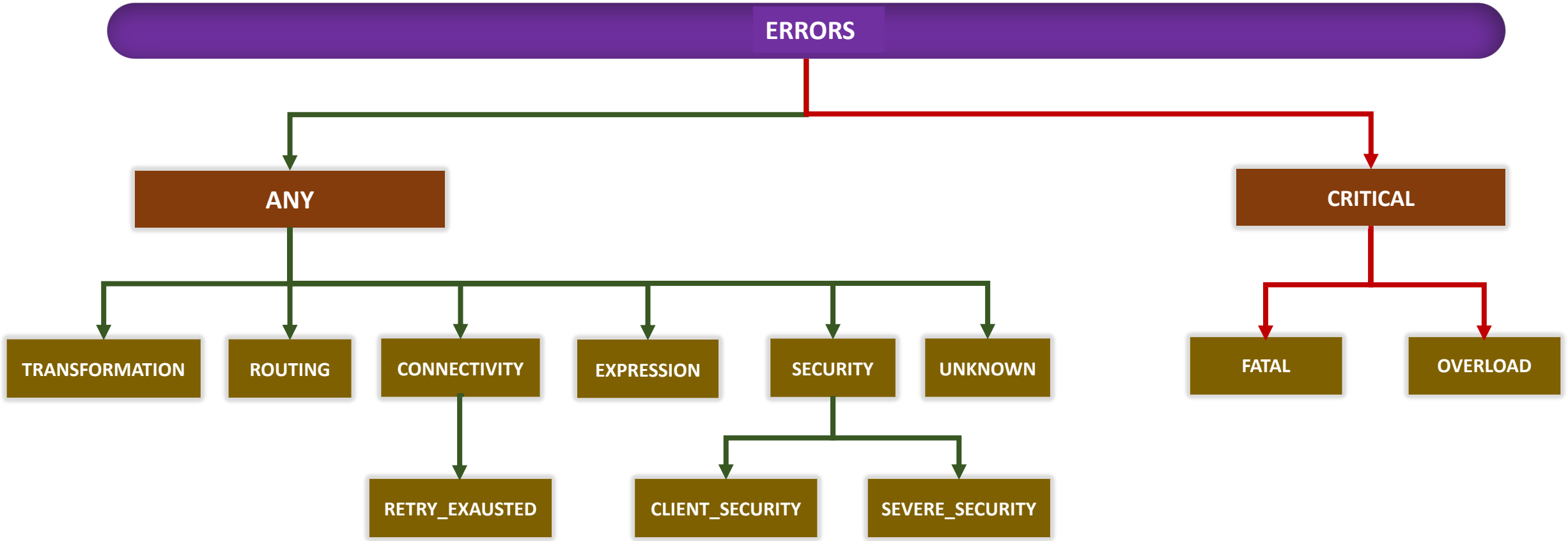
JSON/XML fields: Any on of below

- capital camel case: StreetAdress
- kebab case: street-address
- snake case: sreet_address
- camel case: streetAddress

Variable names: camel case (ex: covidCasePayload)

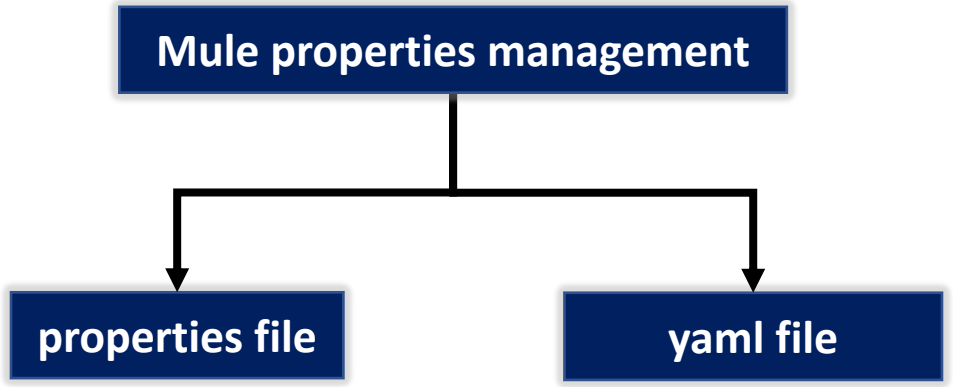
Connector names: Firstletter capital word with space b/w words (ex: Transform Message)

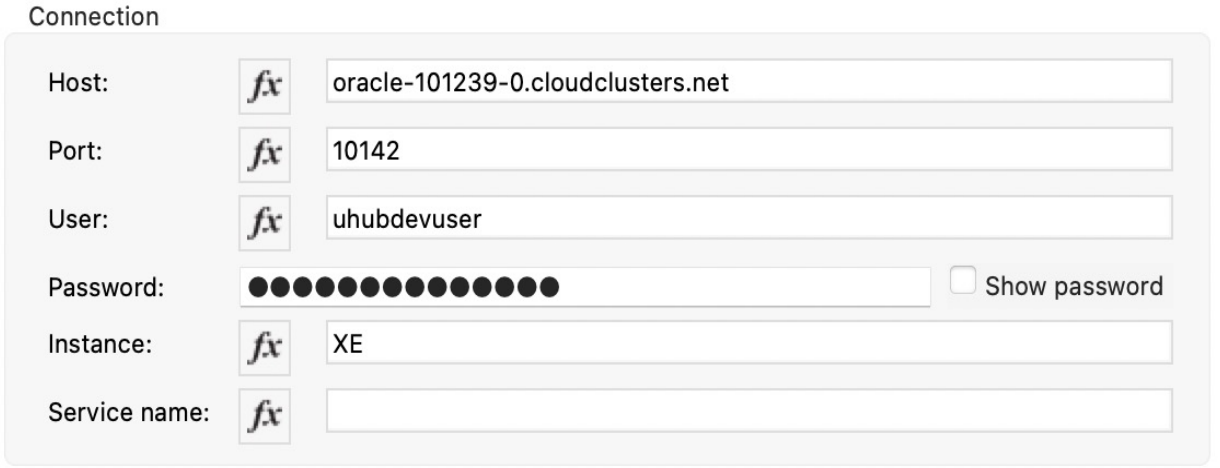
Mule error types



Mule properties management

Properties management : Externalizing properties from mule configuration help us in better code management and need not touch applications on configurations property changes.





1. Create property file or yaml file in src/main/resources.
2. Create “configuration properties” global element.
3. Use ${key} expression extract the property from property file or yaml file.
4. Use Mule::p(‘key’) dataweave script to extract in dataweave.

Mule properties – Secure configuration properties

Properties management : Externalizing properties from mule configuration help us in better code management and need not touch applications on configurations property changes.

1. Create property file or yaml file in src/main/resources.

2. Encrypt password property values:

   - Choose 16 digits encryption key : abcdef0123456789
   - Use algorithm and mode to encrypt password using above key.

3. Create secure configuration property global element.

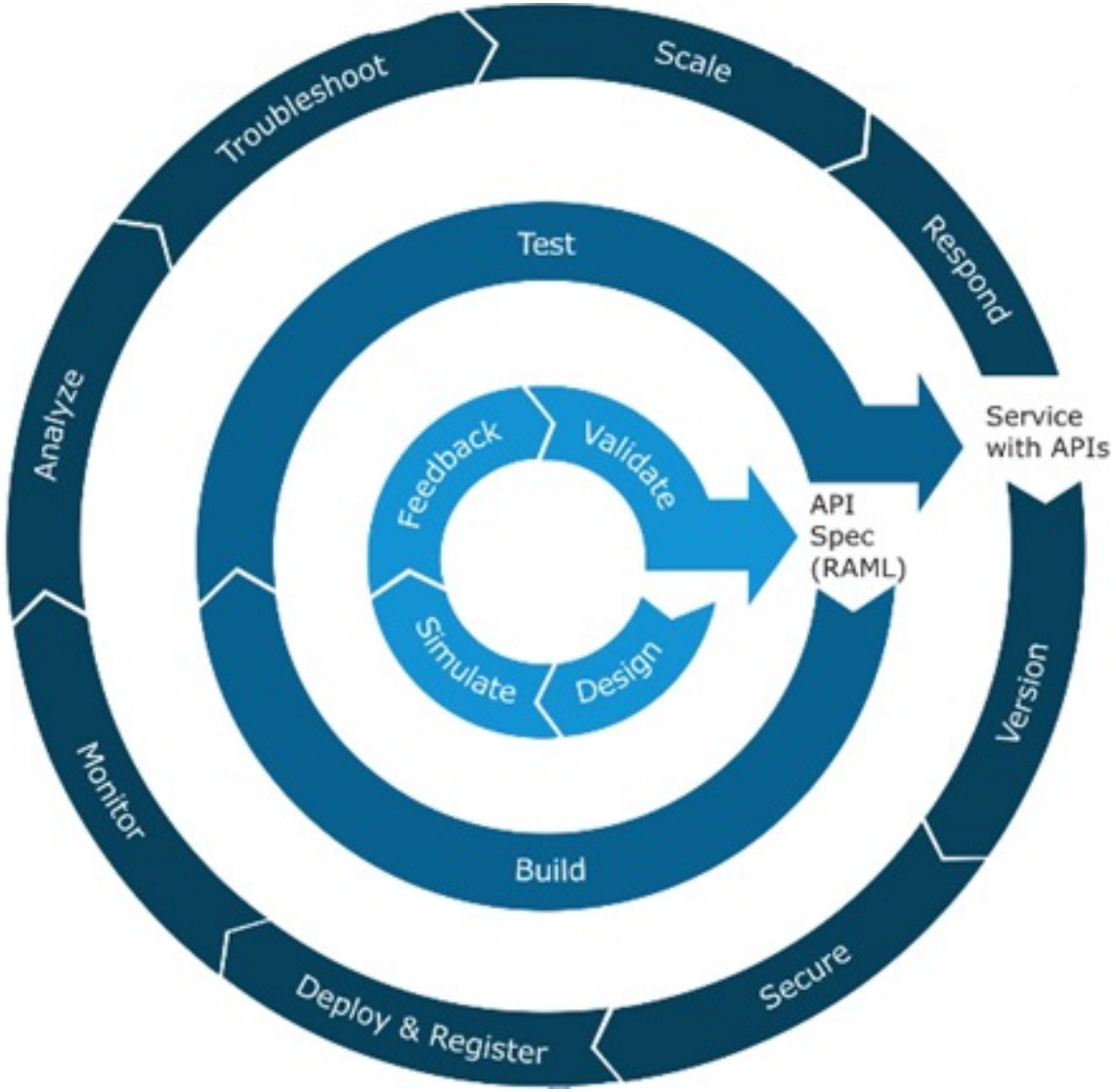
1. Use ${secure::key} expression extract the property from property file or yaml file.

2. Use Mule::p('secure::key') dataweave script to extract in dataweave.

```yaml
http:
  port: "8081"

db:
  uhub:
    host: "oracle-101239-0.cloudclusters.net"
    port: "10142"
    username: "uhubdevuser"
    password: "uhubdevuser123"
    service: "XE"
```

Mule API life cycle



DESIGN

↓

DEVELOPMENT

↓

MANUAL UNIT TESTING

↓

MUNIT TESTING

↓

MANUAL DEPLOYMENT

↓

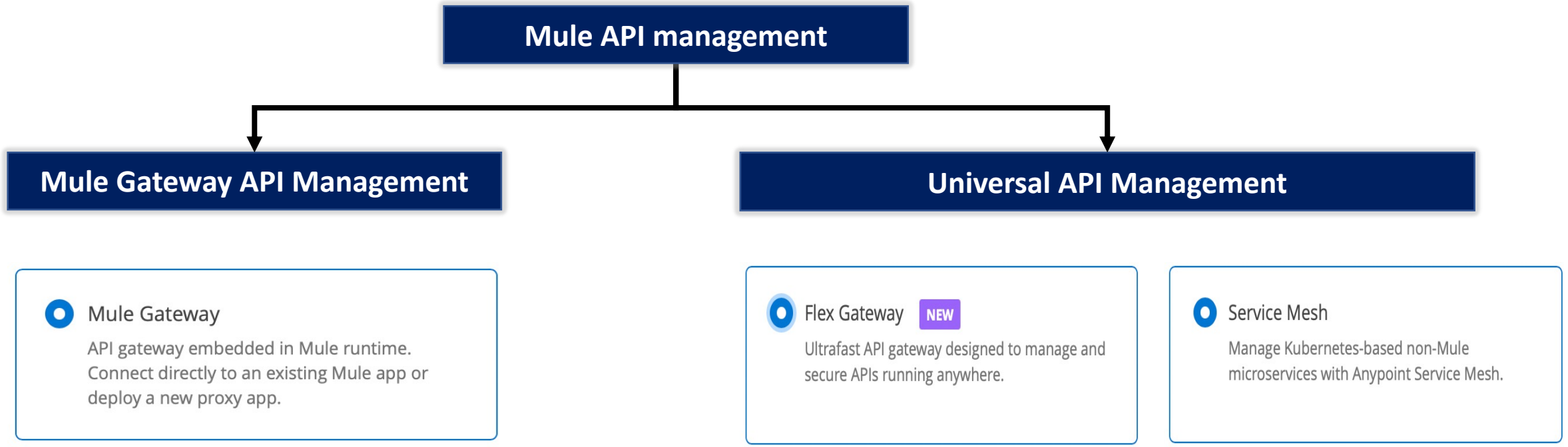
CICD DEPLOYMENT

↓

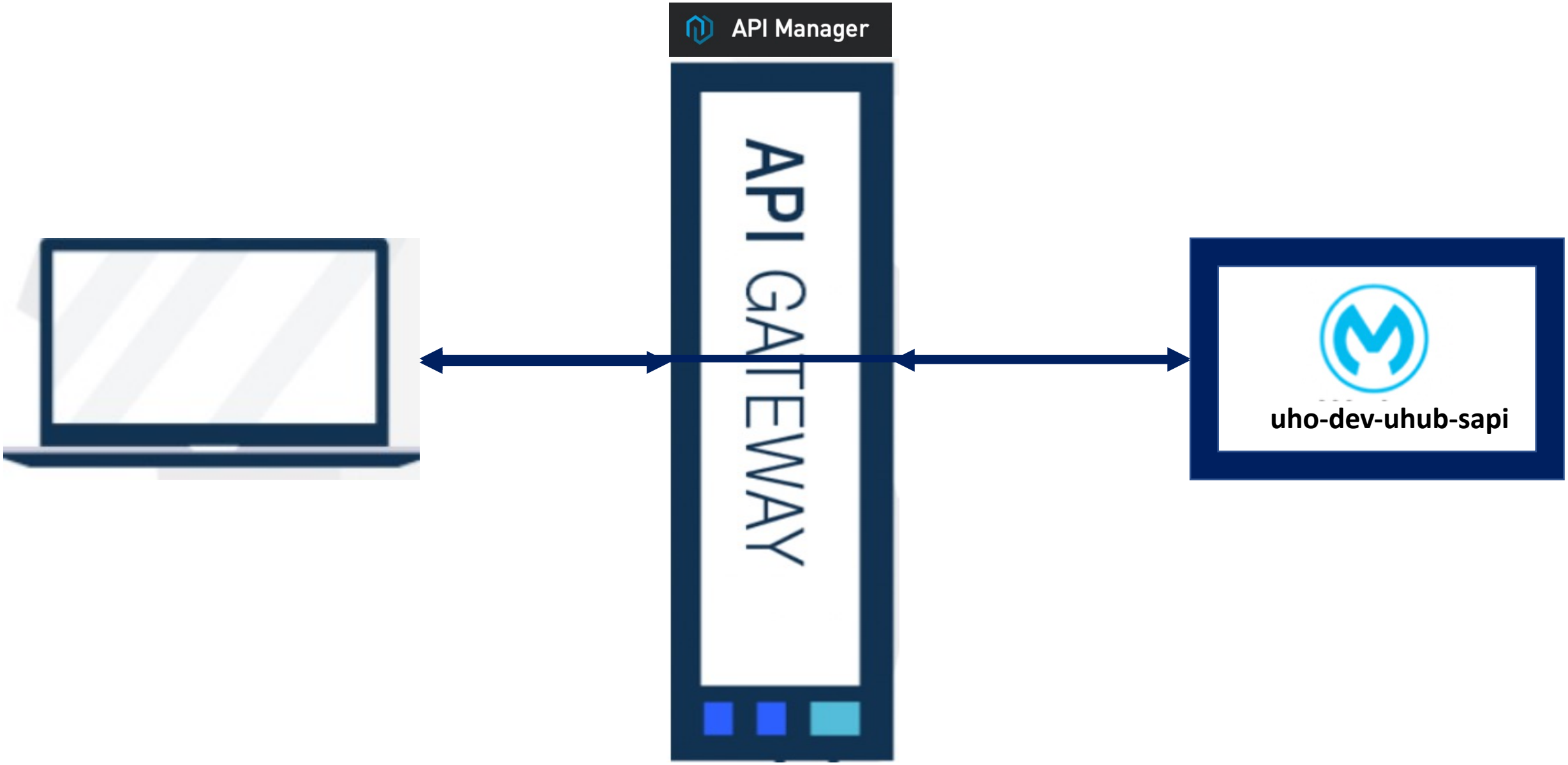
API MANAGEMENT

Mule API management

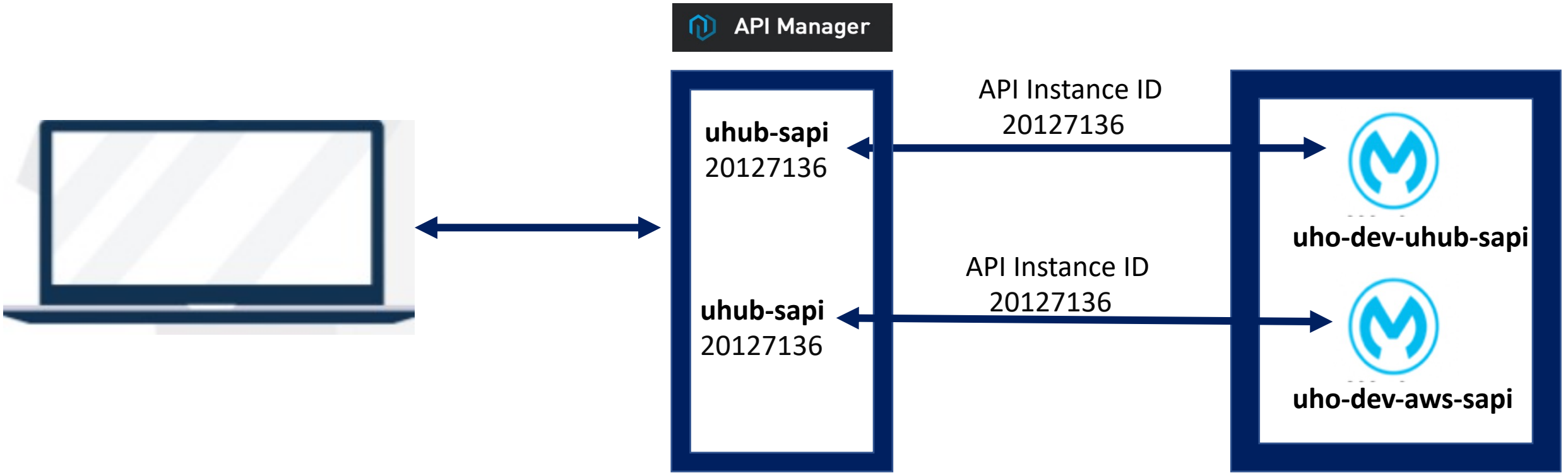
API management : API Management is the process of designing, publishing, documenting, analyzing, versioning, securing and managing API users, traffic, SLAs.



Mule API management



Auto discovery



1. Create auto discovery global element using API instance id.

2. Provide below runtime properties:

   - Target environment client_id (anypoint.platform.client_id)
   - Target environment client_secret (anypoint.platform.client_secret)

Client id enforcement policy

The Client ID Enforcement policy restricts access to a protected resource by allowing requests only from registered client applications. The policy ensures that the client credentials sent on each request have been approved to consume the API.

1. Register client app on API in exchange.

2. Specifies from where in the request to extract the values:

    - HTTP Basic Authentication Header: Requires credentials as part of the authorization header. The application consuming the API must use the basic authentication scheme to send the credentials in the requests.

    - Custom Expression: Accepts an expression each for client_id and client_secret in. the headers, indicating where to extract the credentials from the request.

Rate limiting policy

The Rate Limiting policy enables you to limit the number of requests that an API can accept within a time window. The API rejects any request that exceeds this limit. You can configure multiple limits with window sizes ranging from milliseconds to years.

1. Apply configuration to all API method & resources

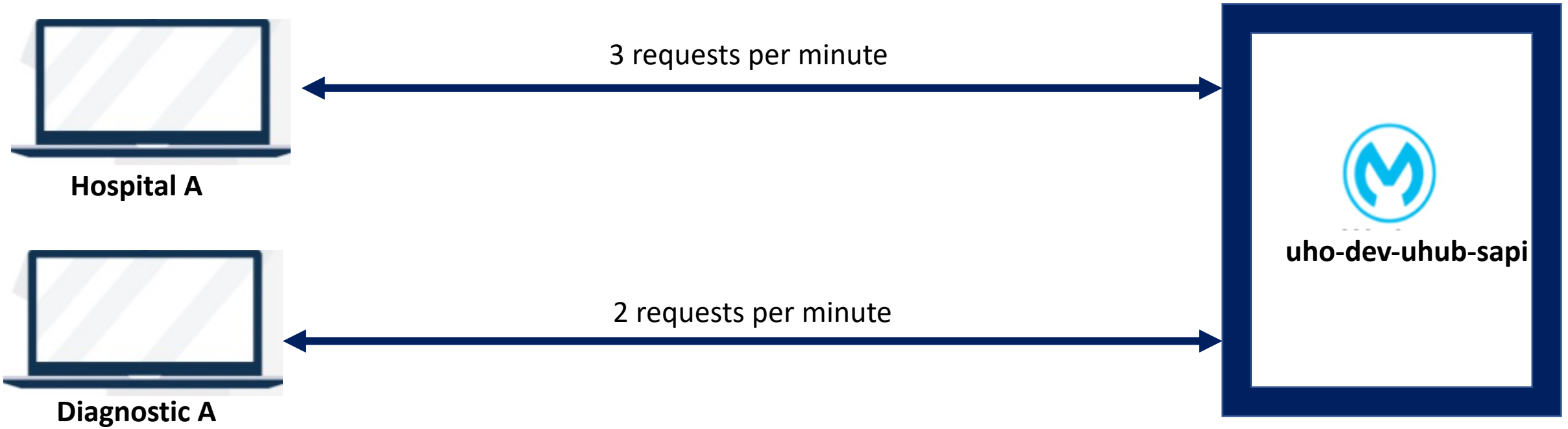
2. Apply configuration to specific API method & resources

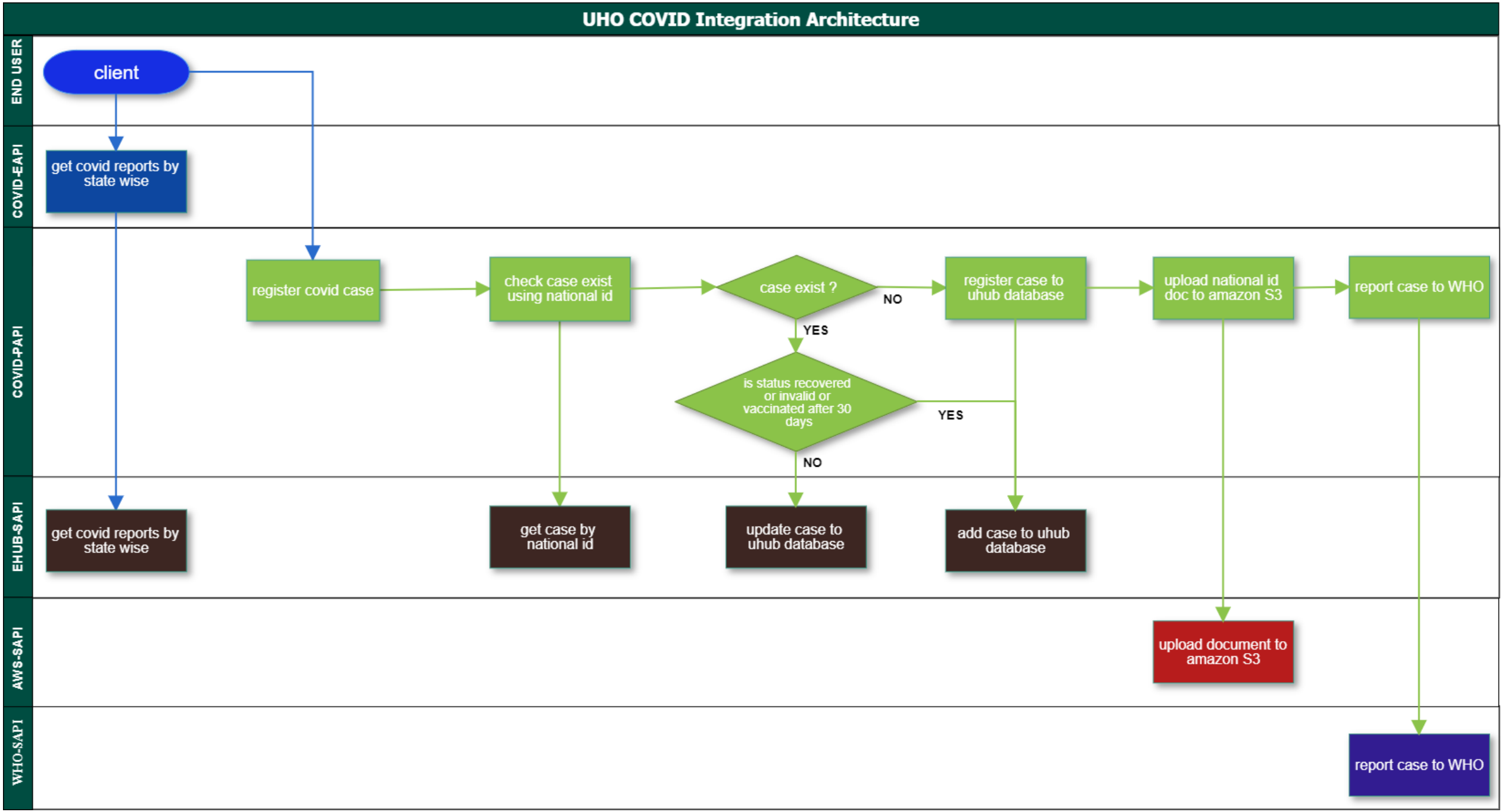
Rate limiting – SLA based policy

The Rate Limiting policy is combination of rate limiting and client id enforcement that enables you to limit the number of requests specified by the level of access granted to the requesting application. The API rejects any request that exceeds this limit. You can configure multiple limits with window sizes ranging from milliseconds to years.

1. Apply configuration to all API method & resources

2. Apply configuration to specific API method & resources




UHO COVID Integration Architecture
END USER
client
COVID-EAPI
get covid reports by state wise
COVID-PAPI
register covid case
check case exist using national id
case exist ?
NO
YES
register case to uhub database
upload national id doc to amazon S3
report case to WHO
is status recovered or invalid or vaccinated after 30 days
YES
NO
EHUB-SAPI
get covid reports by state wise
get case by national id
update case to uhub database
add case to uhub database
AWS-SAPI
upload document to amazon S3
WHO-SAPI
report case to WHO

WSDL elements

<definitions>

<types>
 data type definitions...
</types>

<message>
 definition of the data being communicated...
</message>

<portType>
 set of operations...
</portType>

<binding>
 protocol and data format specification...
</binding>

<service>
 Service URL location...
</service>

</definitions>

Mathematical Operators

DataWeave supports the most common mathematical operators:

Operator	Description
+	For addition.
-	For subtraction.
*	For multiplication.
/	For division.

Equality and Relational Operators

DataWeave supports the following equality and relational operators:

Operator	Description
<	For less than.
>	For greater than.
<=	For less than or equal to.
>=	For greater than or equal to.
==	For equal to.
~=	Equality operator that tries to coerce one value to the type of the other when the types are different.

Logical Operators

DataWeave supports the following logical operators:

Operator	Description
`not`	Negates the result of the input.
`!`	Negates the result of the input.
`and`	Returns true if the result of all inputs is true, false if not.
`or`	Returns true if the result of any input is true, false if not.

Prepend, Append, and Remove Operators

DataWeave supports operators for appending and prepending items within an array:

Operator	Description
>>	Prepends data on the left-hand side of the operator to items in the array on the right-hand side. For example, 1 >> [2] results in [1, 2], prepending 1 to 2 in the array.
<<	Appends data on the right-hand side of the operator to items in the array on the left-hand side. For example, [1] << 2 results in [1, 2], appending 2 to 1 in the array.
+	Appends data on the right-hand side of the operator to items in the array on the left-hand side. For example, [1] + 2 results in [1, 2], appending 2 to 1 in the array. The array is always on the left-hand side of the operator.
-	Removes a specified element of any supported type from an array.

DataWeave control statements

Conditional statement	Description
If else	An if statement evaluates a conditional expression and returns the value under the if only if the conditional expression returns true. Otherwise, it returns the expression under else. Every if expression must have a matching else expression.
Else if	You can chain several else expressions together within an if-else construct by incorporating else if.

DataWeave set default values

Conditional statement	Description
default	Default is used to default the values on fields when the values are empty. It works as if-else internally. Ex: "userName": payload.name default "Undefined"

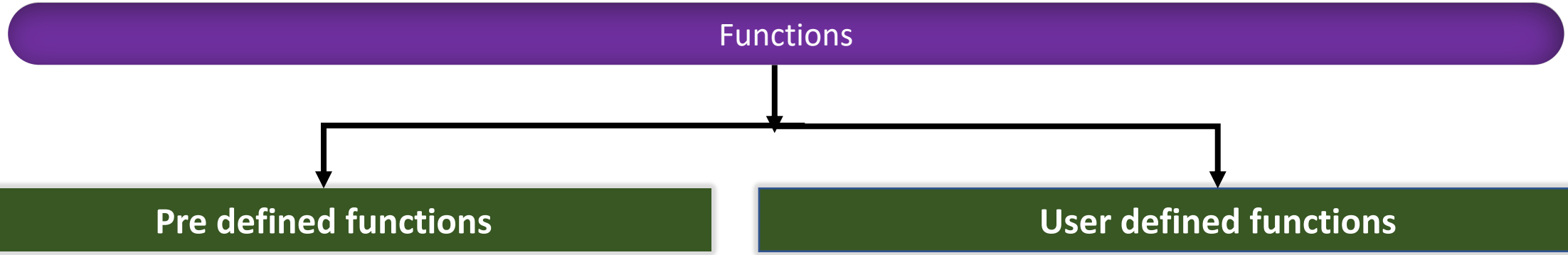
DataWeave pattern matching

Pattern matching	Description

match	The match statement behaves like a match or switch statement in other languages, like Java or C++, and routes an input expression to a particular output expression based on some conditions.
	
	Syntax:
	inputExpression match {
	case <condition> -> <routing expression>
	case <condition> -> <routing expression>
	else -> <default routing expression>
	}

Types of functions

A function allows or provides piece of reusable logic to perform an operation



dw::Core
dw::core::Arrays
dw::core::Binaries
dw::core::Dates
dw::core::Numbers
dw::core::Objects
dw::core::Periods
dw::core::Strings
dw::core::Types
dw::core::URL
dw::Crypto
dw::extension::DataFormat
dw::module::Mime
dw::module::Multipart
dw::Mule
dw::Runtime
dw::System
dw::util::Coercions
dw::util::Diff
dw::util::Math
dw::util::Timer
dw::util::Tree
dw::util::Values
dw::xml::Dtd

Syntax: fun myFunction(param1, param2, ...) = <code to execute>

Syntax: fun myFunction(param1: Type, param2: Type): ResultType = <code to execute>

do statement

Operator	Description

do	A do statement creates a scope in which new variables, functions, annotations, or namespaces can be declared and used. The syntax is similar to a mapping in that it is composed of a header and body separated by ---. Its header is where all the declarations are defined, and its body is the result of the expression.
	
	Syntax:
	do {
	  header
	  --- 	
	  body
	}

DataWeave core math functions

dw::Core math functions	Description
abs	Returns the absolute value of a number
ceil	Rounds a number up to the nearest whole number
floor	Rounds a number down to the nearest whole number
round	Rounds a number up or down to the nearest whole number
min	Returns the lowest Comparable value in an array
minBy	Iterates over an array of objects to return the lowest value of comparable elements from it
max	Returns the highest Comparable value in an array
maxBy	Iterates over an array of objects to return the highest value of comparable elements from it
isEven	Returns true if the given number is an integer (which lacks decimals), false if not
isOdd	Returns true if the number or numeric result of a mathematical operation is odd, false if not
isInteger	Returns true if the given number is an integer (which lacks decimals), false if not
isDecimal	Returns true if the given number contains a decimal, false if not

DataWeave core string functions

dw::Core string functions	Description
contains	Returns true if an input contains a given value, false if not.
Find	Returns indices of an input that match a specified value.
lower	Returns the provided string in lowercase characters.
upper	Returns the provided string in uppercase characters.
matches	Checks if an expression matches the entire input string.
replace	Performs string replacement.
splitBy	Splits a string into a string array based on a value that matches part of that string
startsWith	Returns true or false depending on whether the input string starts with a matching prefix.
endsWith	Returns true if a string ends with a provided substring, false if not.
Trim	Removes any blank spaces from the beginning and end of a string
isEmpty	Returns true if the given input value is empty, false if not.
isBlank	Returns true if the given string is empty (""), completely composed of whitespaces, or null. Otherwise, the function returns false.

DataWeave core “Of” functions

dw::Core	Description
sizeOf	Returns the number of elements in an array. It returns 0 if the array is empty.
typeOf	Returns the primitive data type of a value, such as String
namesOf	Returns an array of strings with the names of all the keys within the given object.
keysOf	Returns an array of keys from key-value pairs within the input object.
valuesOf	Returns an array of the values from key-value pairs in an object.
indexOf	Returns the index of the *first* occurrence of the specified element in this array, or -1 if this list does not contain the element.
lastIndexOf	Returns the index of the *last* occurrence of the specified element in a given array or -1 if the array does not contain the element.
entriesOf	Returns an array of key-value pairs that describe the key, value, and any attributes in the input object.

DataWeave core map function

dw::Core	Description

map	Iterates over items in an array and outputs the results into a new array.
	
	Syntax-1: Without parameters:
	
	arrayPayload map {
	    ($$): $
	}
	
	Syntax-2: With parameters:
	
	arrayPayload map(value, index) -> {
	    (index): value
	}

DataWeave core mapObject function

dw::Core	Description

mapObject	Iterates over an object using a mapper that acts on keys, values, or indices of that object.
	
	Syntax-1: Without parameters:
	
	objectPayload mapObject {
	($$$) : { ($):$$}
	}
	
	Syntax-2: With parameters:
	
	objectPayload mapObject (value, key, index) -> {
	(key): value
	}

DataWeave core filter function

dw::Core	Description
filter	Iterates over an array and applies an expression that returns matching values. Syntax-1: Without parameters: arrayPayload filter(condition) $ represents the current value on each iteration Syntax-2: With parameters: arrayPayload filter (value, index) -> condition

DataWeave core filterObject function

dw::Core	Description
filterObject	Iterates a list of key-value pairs in an object and applies an expression that returns only matching objects, filtering out the rest from the output. Syntax-1: Without parameters: objectPayload filterObject(condition) Syntax-2: With parameters: objectPayload filterObject (key, value, index) -> condition

DataWeave core flatten & flatMap functions

dw::Core	Description
flatten	Turns a set of subarrays (such as ([1,2,3], [4,5]) into a single array [1,2,3,4,5]. Syntax: flatten(arrayOfArraysPayload)

flatMap	Iterates over each item in an array and flattens the results.
	
	Syntax-1: Without parameters:
	
	arrayOfArraysPayload flatMap {
	(($$): $
	}
	
	Syntax-2: With parameters:
	
	arrayOfArraysPayload flatMap(value, index) -> {
	(index): value
	}

DataWeave core by functions

dw::Core	Description
distinctBy	Iterates over the input and returns the unique elements in it. Syntax-1: arrayPayload distinctBy $ Syntax-2: arrayPayload distinctBy(value) -> value
orderBy	Reorders the elements of an input using criteria that acts on selected elements of that input. Syntax-1: arrayPayload orderBy $ Syntax-2: arrayPayload orderBy(value) -> value
groupBy	Returns an object that groups items from an array based on specified criteria. Syntax-1: arrayPayload groupBy $ Syntax-2: arrayPayload groupBy(value) -> value
joinBy	Merges an array into a single string value and uses the provided string as a separator between each item in the list. Note that joinBy performs the opposite task of splitBy. Syntax: arrayPayload joinBy “separator”

DataWeave core reduce function

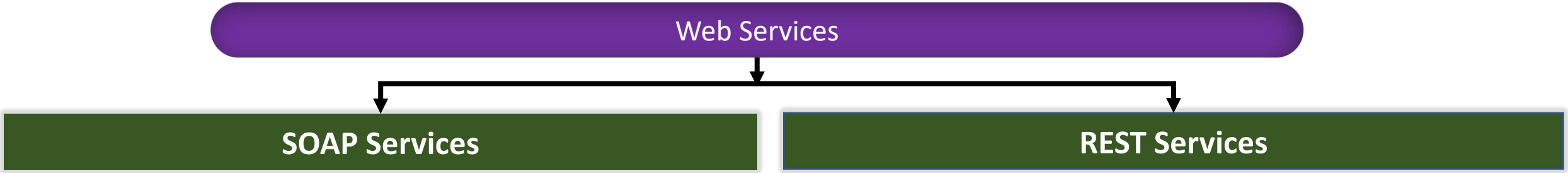
dw::Core	Description
reduce	Applies a reduction expression to the elements in an array. Syntax1: arrayOfPayload reduce($, $$) Example: [2,3,4] reduce($ + $$) Syntax-2: arrayOfPayload reduce(item, acc) -> item + acc Example: [2,3,4] reduce(item, acc) -> item + acc **Explanation:** Iteration 1: item is 2 (assigns first item to acc = 2) Iteration 2: item is 3 & acc is 2 -> 3 + 2. (assigns result to acc = 5) Iteration 3: item is 4 & acc is 5 -> 4 + 5. (assigns result to acc = 9) Result: 9

DataWeave core other functions

dw::Core	Description
++	This statement ++ concatenates the string with string, adds objects together into a single object. Adds two arrays into into a new array.
--	The statement -- removes all instances of the specified items from an array. Other versions act on objects.
pluck	It iterates over an object and returns an array of keys, values, or indices from the object.
to	Returns a range with the specified boundaries.
uuid	Returns a v4 UUID using random numbers as the source.
zip	Merges elements from two arrays into an array of arrays.
unzip	Performs the opposite of zip. It takes an array of arrays as input.

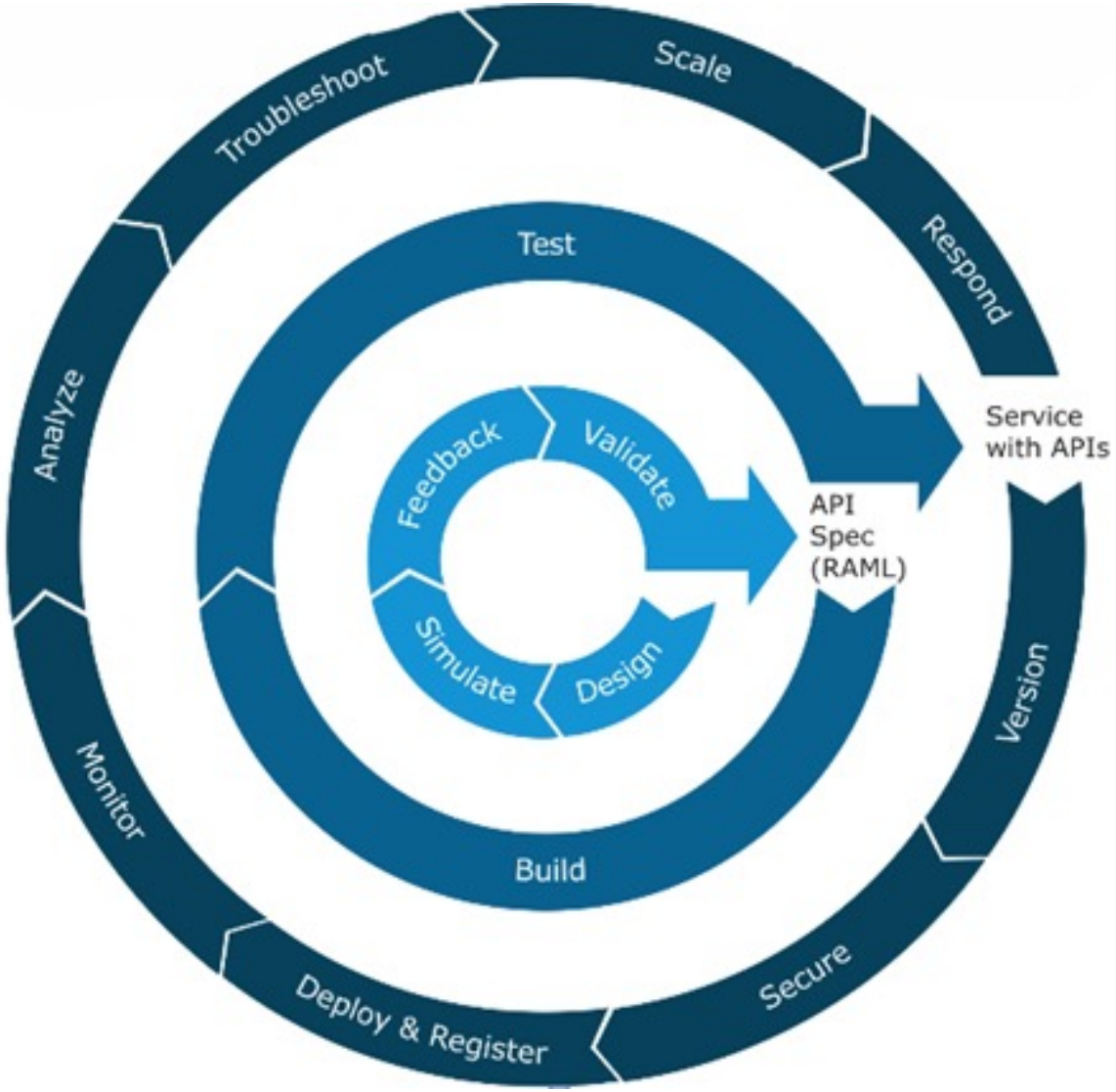
Web service

Web service : A web service is a program which runs in server and whose results are sharable over network/integration to make data transfer between applications.



• SOAP services are legacy approach, and the development of SOAP services is very less.	• REST services are latest and called as REST API's.
• SOAP uses XML for all messages.	• REST supports many messages mime types (JSON, XML etc.)
• Designing of SOAP API uses WSDL.	• Designing of REST uses OAS, RAML

Mule API life cycle



DESIGN

↓

DEVELOPMENT

↓

MANUAL UNIT TESTING

↓

MUNIT TESTING

↓

MANUAL DEPLOYMENT

↓

CICD DEPLOYMENT

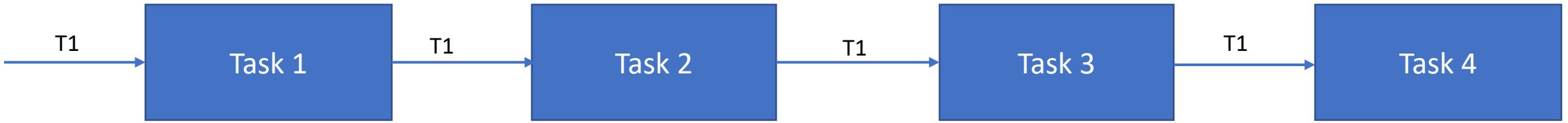
↓

API MANAGEMENT

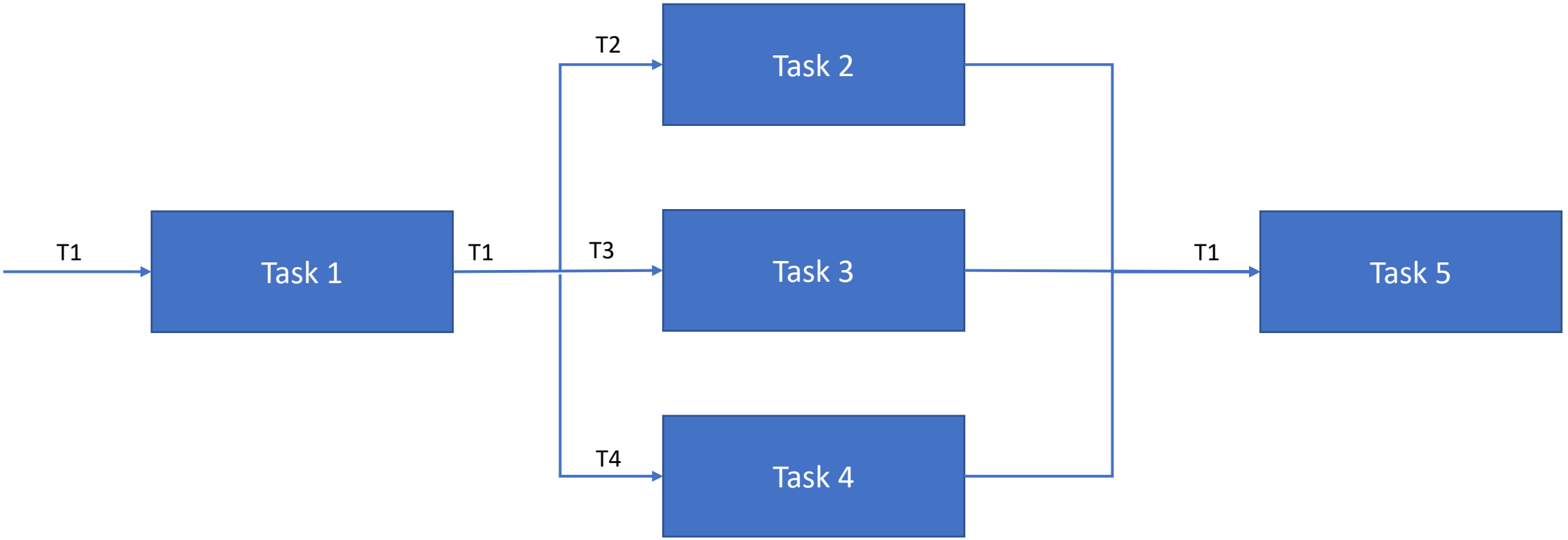
Salesforce integration

- Setup developer salesforce account.
- Create a custom object emp_profile with fields emp_phone, emp_email, emp_skills.
- Work with salesforce create integration.
- Work with salesforce query operation to fetch object.
- Work with salesforce streaming.

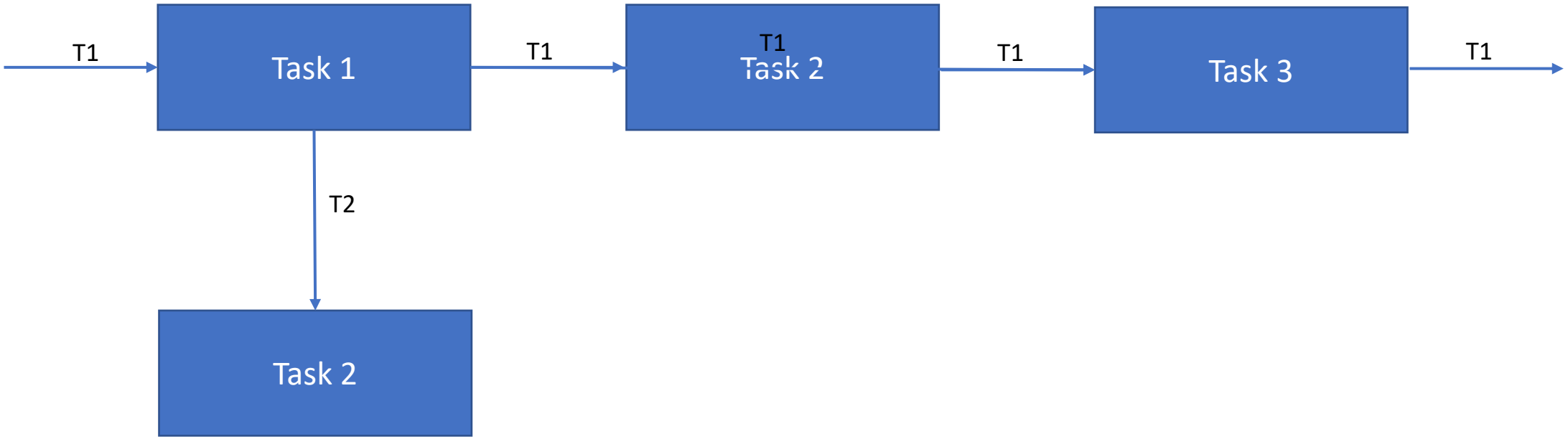
Sequential processing



Parallel processing



Asynch processing



THANK YOU

HAPPY LEARNING ☺