

# MULESOFT

---

*SAI KIRAN YALAKALA*

# Index

Introduction about  
Mulesoft , API &  
Postman

API-LED Connectivity

Anypoint Platform  
(Design Center,  
Exchange, Runtime  
manager & API  
Manager)

Mule4 Structure (Set  
Payload, Attributes &  
Set Variable )

Anypoint Studio  
(Package Explorer,  
Canvas, Palette &  
Console)

Flows (Flow, Subflow &  
Private Flow)

Flow reference

Connectors (FTP, SFTP,  
MySQL, Salesforce &  
AWS)

Automated & Manual  
Project

Transform Message

Configuration  
properties

DataWeave ()

Choice Reutter

Scatter Gather

Error Handling (On-  
error Continue, On  
Error Propagation &  
Raise Error)

# What is Mulesoft

Drag and Drop tool

Integration platform

Business data connector  
(Communicates between 2 or more systems)

Acquired by Salesforce on May'2018

Founded in 2006 by Ross Mason

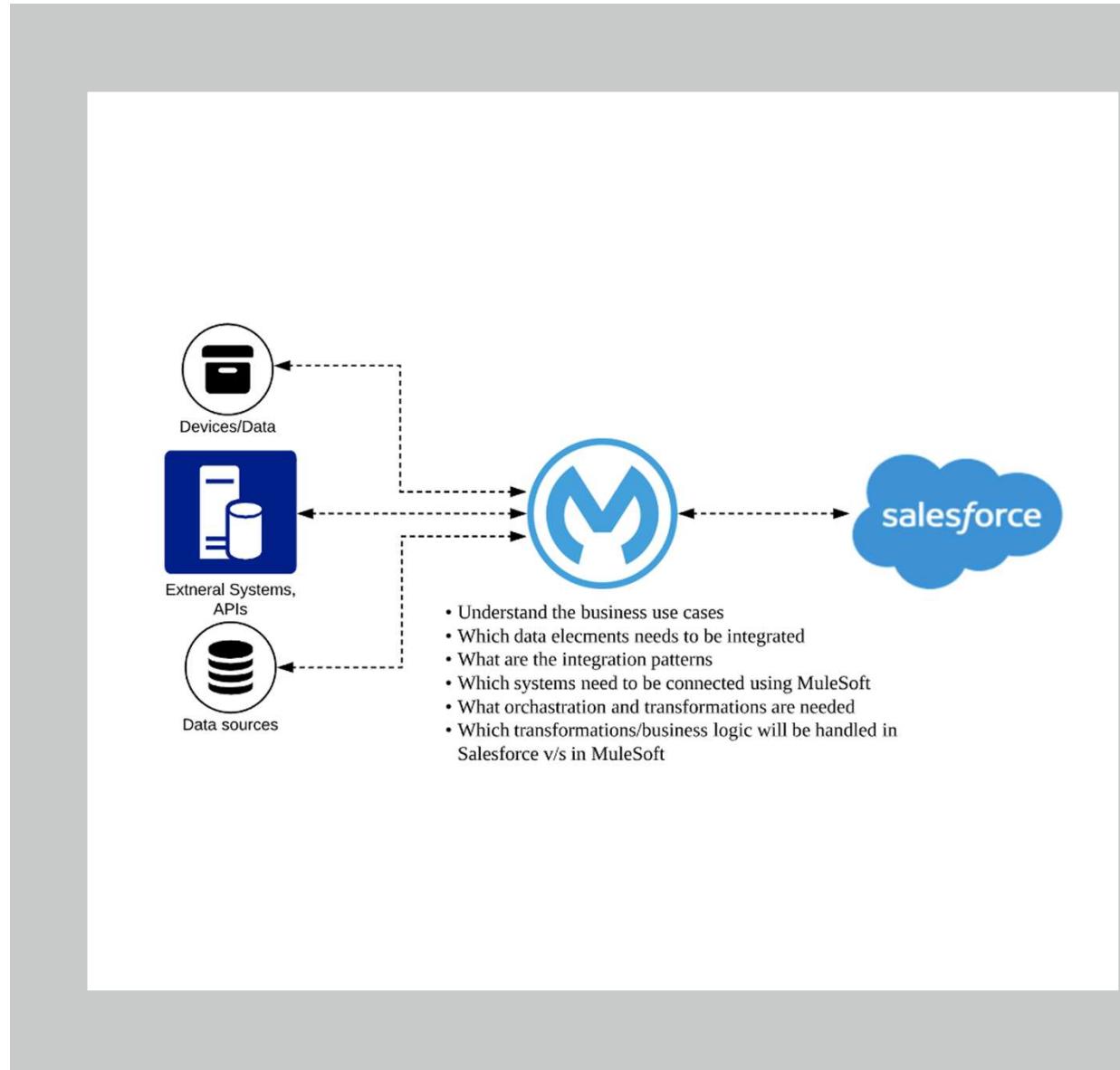
Mule = Donkey + Horse

Competitors : DELL BOOMI, TIBCO etc...

EAI Product

# What is Integration

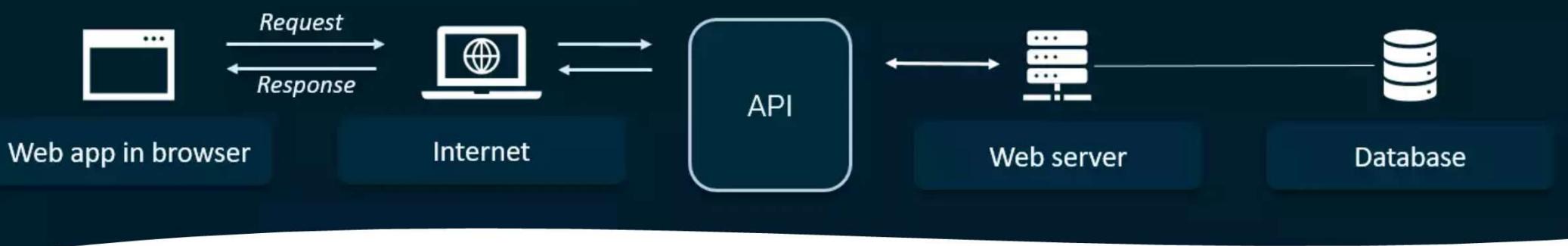
- Integration is the act of bringing together smaller components into a single system that functions as one
- Integration is the process of connecting multiple enterprise systems, devices, or applications to work as a whole
- Milk + Water + Sugar+Tea powder = Tea



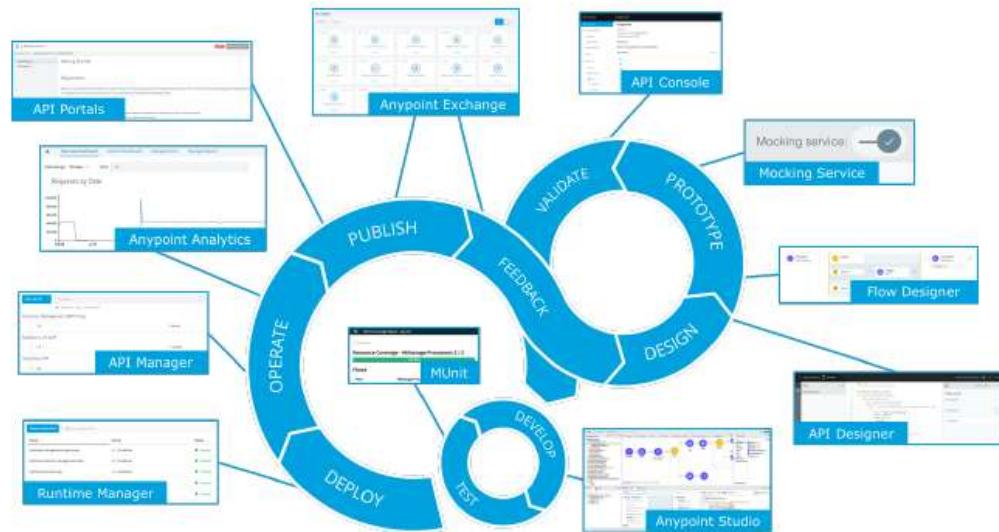
# What is an API

- An **API** is a set of programming code that enables data transmission between one software product and another. It also contains the terms of this data exchange.
- Reusable piece of code
- Divided into 3 Layers
  - System API
  - Process API
  - Experience API

## How API works



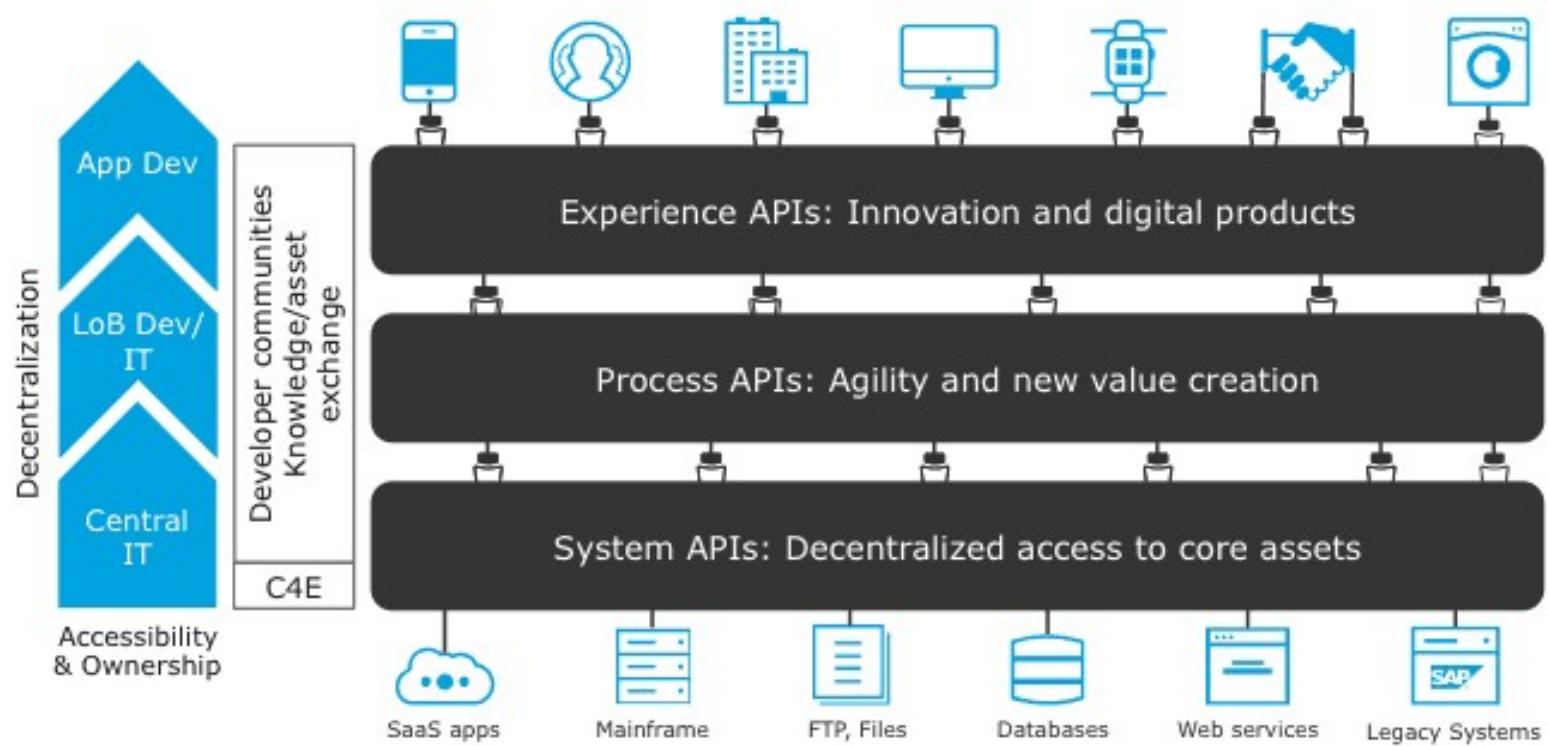
# What is an API specification



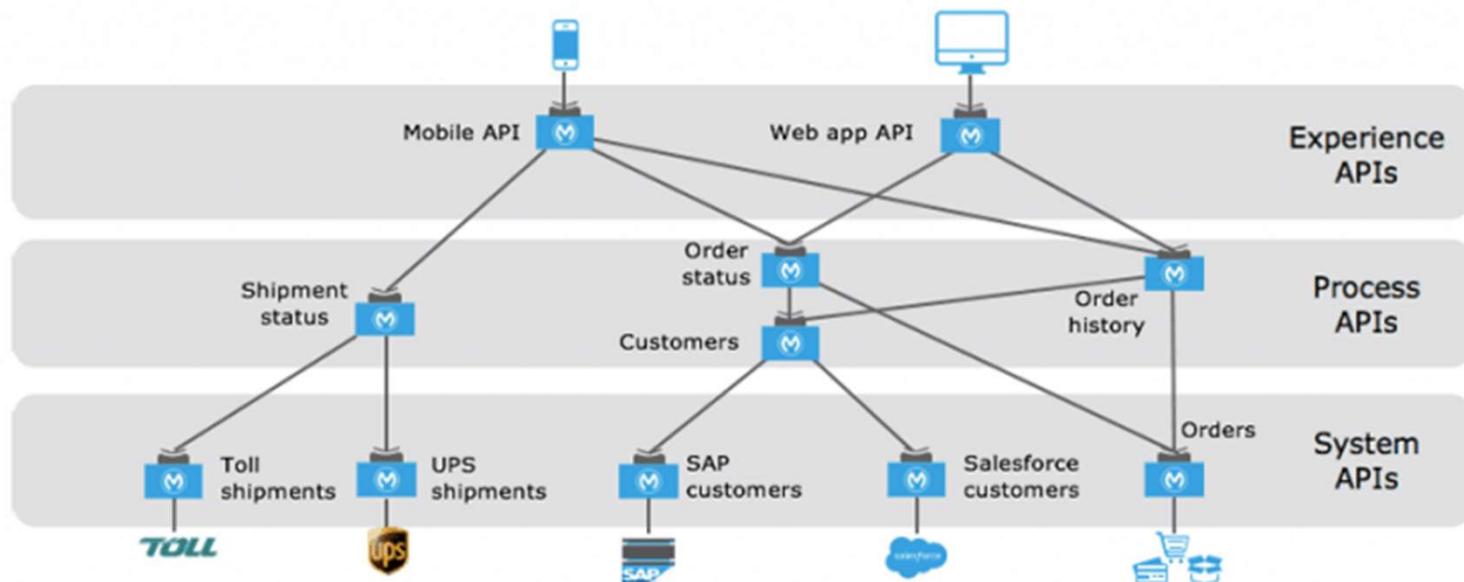
## API Life Cycle

- API Specification is nothing but how an API works
  - What kind of request it accepts
  - What kind of Response it returns
  - Mandatory fields
  - Schema validation (Datatypes)
  - Reliability pattern

# API LED Connectivity



# Architecture



# API LED Connectivity – Real Time

- Manager need an employee details where employee id is 888777
- Logic: Browser->EmployeeID :888777-> Should be 6 digits ->DB
  - Response: {  
    “name”: “Saikiran”,  
    “Skill”: “MuleSoft”  
    “Lcation”: Hyderabad  
}
- Manager SAPI: Database logic (Select \* from employees where employeeId=attributes.queryparams.id)
- Manager PAPI : Business logic (Should be 6 digits/Should not be empty/Only Numbers)
- Manager EAPI/XAPI: URL expose

Government  
SAPI : Used to  
fetch raw  
details from  
end system

```
government-sapi/master ▾

1  #%RAML 1.0
2  title: government-sapi
3
4  traits:
5    client-id-required:
6      headers:
7        client_id:
8          type: string
9        client_secret:
10       type: string
11      responses:
12        401:
13          description: Unauthorized
14        500:
15          description: Internal server error
16
17 /government:
18   description: fetching the age details
19   get:
20     is:
21       - client-id-required
22     responses:
--  ---
```

# ClientID & ClientSecret Enforcement

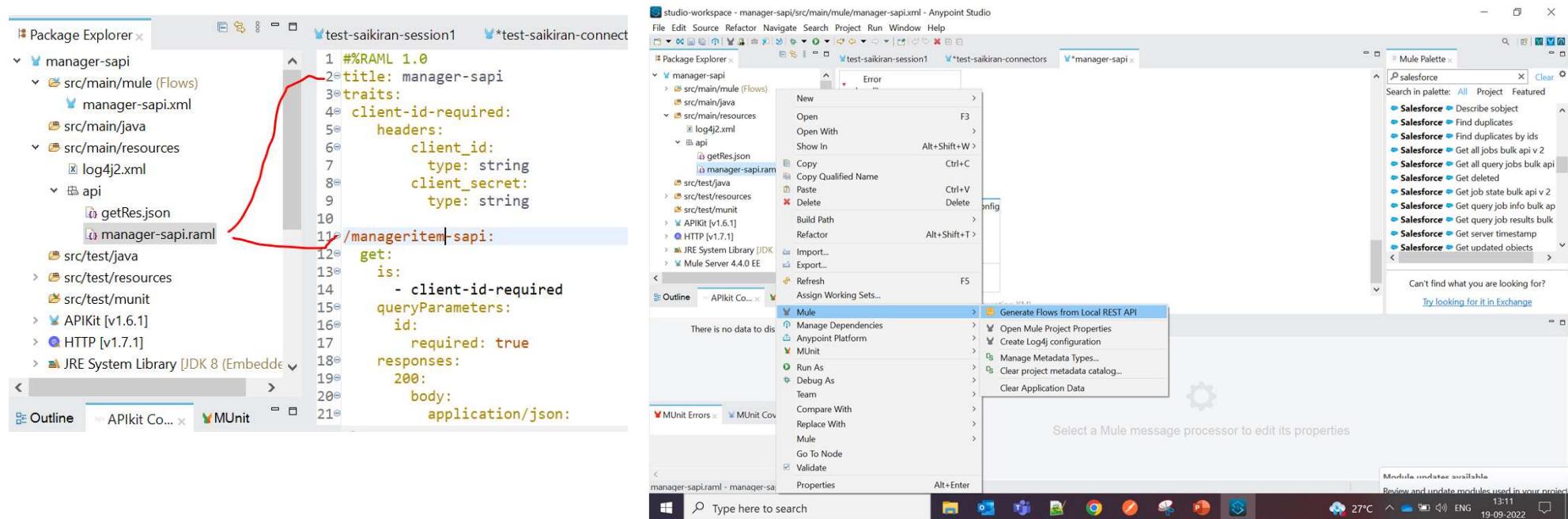
```
manager-sapi/master

1  #%%RAML 1.0
2  title: manager-sapi
3  traits:
4    client-id-required:
5      headers:
6        client_id:
7          type: string
8        client_secret:
9          type: string
10
11 /manager-sapi:
12   get:
13     is:
14       - client-id-required
15     queryParameters:
16       id:
17         required: true
18     responses:
19       200:
20         body:
21           application/json:
22             example: !include getRes.json
```

- Traits: To enforce the policies
  - Defining Traits: Go to API manager -> Policies -> Get ClientID & ClientSecret Policies
  - <https://docs.mulesoft.com/api-manager/2.x/prepare-raml-task>

# Publish to Exchange & Import to Studio, Changing Resource name

- Create a new project & Import the RAML zip file
  - If Manager-sapi api name is changed to manageritem-sapi in **RAML**
  - Goto: src/main/resources -> api ->sapi.raml > generate flow from local rest api
- Change will be deployed – Check once



# Government-PAPI (RAML)

- Business logic is required in PAPI
- The Business logic here is (if Age > 60, “Eligible for pension” else, “NOT”

```
government-papi/master ✘

8   client_secret:
9     type: string
L0   responses:
L1     401:
L2       description: Unauthorized
L3     500:
L4       description: Internal server error
L5
L6 /government-papi:
L7   description: fetching the age details
L8   get:
L9     is:
L10    - client-id-required
L11   queryParameters:
L12     age:
L13       type: integer
L14       required: true
L15   responses:
L16     200:
L17       body:
L18         application/json:
L19           example: !include getresp.json
```

# Rest and SOAP web services

SOAP	REST
1. Simple Object Access Protocol	1. Representational State Transfer
2. Function-driven (data available as services, e.g.: "getUser")	2. Data-driven (data available as resources, e.g. "user").
3. Message format supported: XML	3. Message format supported: Plain text, HTML, XML, JSON, YAML, etc.
4. Transfer Protocols supported: HTTP, SMTP, UDP, etc.	4. Transfer Protocols supported: HTTP
5. SOAP is recommended for Enterprise apps, financial services, payment gateways	5. REST is recommended for mobile applications and Social networking applications.
6. Highly secure and supports distributed environment.	6. Less secured and not suitable for distributed environment.

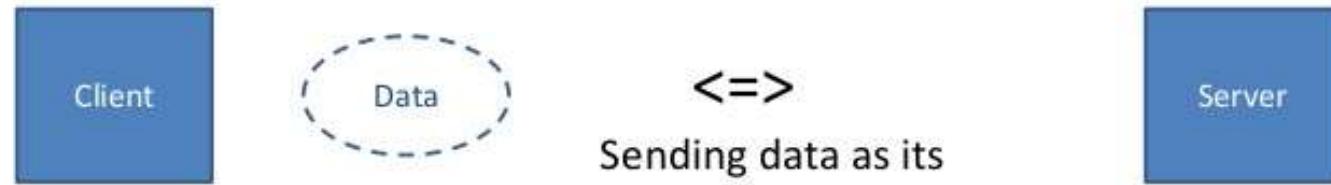
**Fig. SOAP vs REST**

# SOAP vs REST

## SOAP



## REST



\*REST are mostly used in industry

# Invoking Rest API

External Rest API : <https://dummy.restapitexample.com/api/v1/employees>

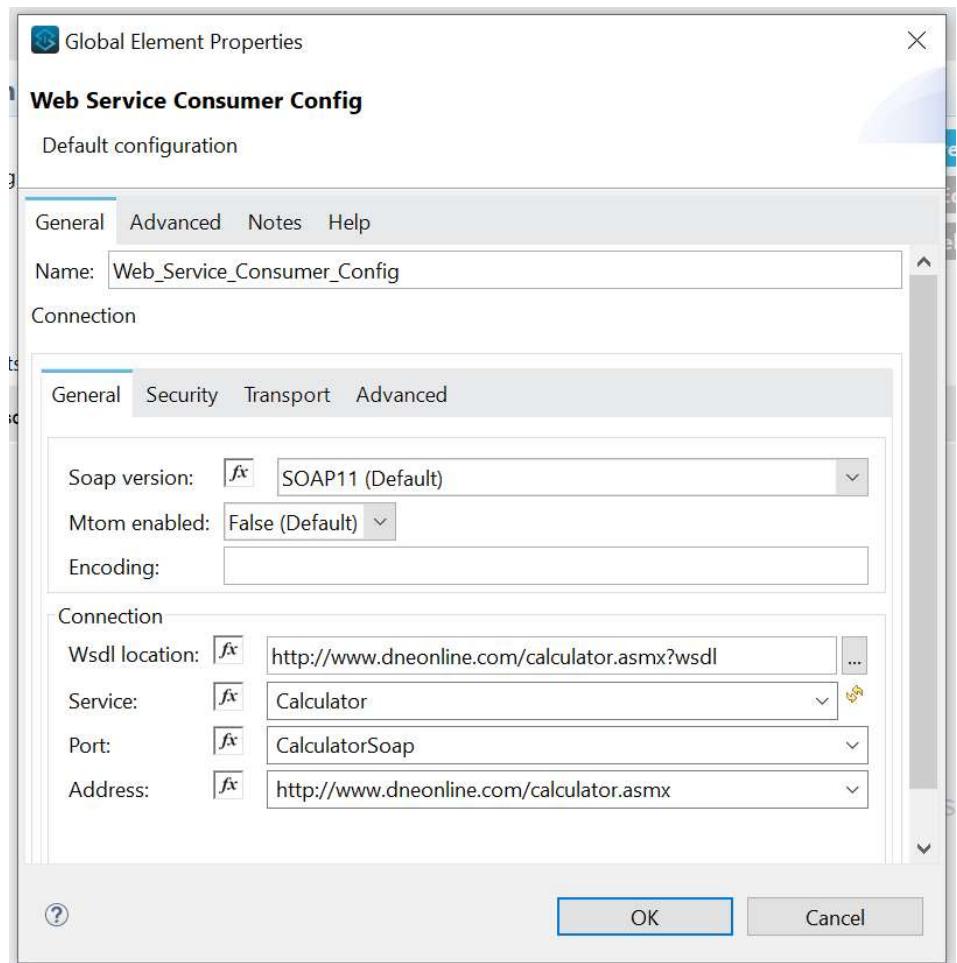
The screenshot shows the Mule Studio interface with a flow named 'dummyrestFlow'. The flow consists of a 'Listener' component followed by a 'Request' component. The 'Request' component has a 'Display Name' of 'Request', a 'Configuration' of 'HTTP\_Request\_configuration' pointing to 'http://dummy.restapitexample.com/api/v1/employees', and a 'Method' of 'GET (Default)'.

The screenshot shows the Postman application interface. A GET request is made to 'http://localhost:8081/restapi'. The response is a JSON object with a 'status' key of 'success' and a 'data' key containing an array of two employee records. The first record has an 'id' of 1, a name of 'Tiger Nixon', a salary of 320800, an age of 61, and an empty profile image. The second record has an 'id' of 2.

```
1 "status": "success",
2 "data": [
3   {
4     "id": 1,
5     "employee_name": "Tiger Nixon",
6     "employee_salary": 320800,
7     "employee_age": 61,
8     "profile_image": ""
9   },
10  {
11    "id": 2,
12    "employee_name": "apseudouser",
13    "employee_salary": 200000,
14    "employee_age": 66,
15    "profile_image": ""
16  }
17 ]
```

# Invoking SOAP API

- SOAP UI website:  
<https://www.soapui.org/docs/soap-and-wsdl/working-with-wsdls/>
- SOAP WSDL Example:  
<http://www.dneonline.com/calculator.asmx?wsdl>
- Setup
  - Connector: Web Service consume
  - Configure with the URL example
  - Service, Port and Address will come by default
  - Use (Add, Multiply, Divide) operator as of requirement



# Soap API : Mapping & Response

soap-apiFlow

```
graph LR; Listener((Listener)) --> S1((Set Variable)); S1 --> S2((Set Variable)); S2 --> TM[Transform Message]; TM --> Consume((Consume));
```

Error handling

Message Flow Global Elements Configuration XML

Transform Message

Input

Payload : Any Define

Variables

- v1 : String
- v2 : String

Attributes : Object

- listenerPath : String
- rawRequestPath : String
- relativePath : String
- maskedRequestPath : String
- version : String
- schema : String

Output

Xml<Add>

Add : Object

- intA : Number
- intB : Number

Sorry, syntax errors prevented the mappings display

Context

Output Payload

```
1 %dw 2.0
2 output application/xml
3 ns ns0 http://tempuri.org/
4 ---
5 {
6     ns0#Add: {
7         ns0#intA: vars.v1 as Number,
8         ns0#intB: vars.v2 as Number
9     }
10 }
```

GET http://localhost:8082/soapui

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (3) Test Results

Pretty Raw Preview Visualize Text

```
1 body:<AddResponse xmlns="http://tempuri.org/"><AddResult>45</AddResult></AddResponse>,
2 headers: [],
3 attachments: []
```

http://localhost:8082/soapui?number1=10&number2=20

GET http://localhost:8082/soapui?number1=10&number2=20

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> number1	10	
<input checked="" type="checkbox"/> number2	20	

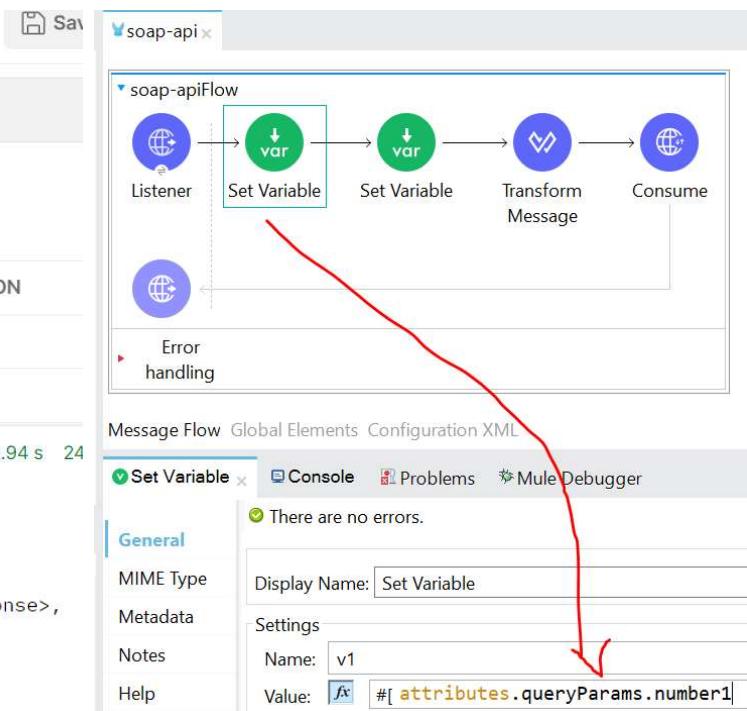
Body Cookies Headers (3) Test Results

Pretty Raw Preview Visualize Text

```

1 <?
2 body:<AddResponse xmlns="http://tempuri.org/"><AddResult>30</AddResult></AddResponse>,
3 headers: [],
4 attachments: []
5 <?

```



# Passing variables in Postman via queryParams but not in value

# How to Print the result in Logger

The screenshot displays the Mule Studio interface with the following components:

- Middle Panel (Message Flow):** Shows a flow named "soap-apiFlow" with the following sequence:
  - Listener
  - Set Variable (with value "var")
  - Set Variable (with value "var")
  - Transform Message
  - Consume
  - Logger
- Middle Panel (Logger Configuration):** A "Logger" component is selected. Its configuration includes:
  - Display Name: Logger
  - Generic Message: `#[ payload.body.AddResponse.AddResult ]`
  - Level: INFO (Default)
  - Category: (empty)
- Right Panel (Mule Palette):** Shows the available components categorized under "Favorites" and "Core".
- Right Panel (SoapUI Test Results):** A SoapUI test run for a GET request to `http://localhost:8082/soapui?number1=500&number2=3` is shown. The log output includes:

```
com.mulesoft.routing.RoutingServiceFactoryBean: Creating Service {http://client.internal.services/processor.LoggerMessageProcessor: 503}
```
- Bottom Right Panel (Body Content):** The response body is displayed in Pretty, Raw, and Preview formats. The raw XML content is:

```
1 <AddResponse>
2   <body><AddResult xmlns="http://tempuri.org/">503</AddResult>
3   <headers> []
4   <attachments> []
5 </AddResponse>
```

# Anypoint Platform



Design Center - Design and develop API's



Anypoint Exchange - Place where we will be able to share, discover api's and reusable assets



API Manager - Helps in Managing API's by invoking policies



Runtime Manager - Where applications are deployed



Access management -Configure access and permissions within organization



Anypoint Monitoring/Visualizer-Provides monitoring and real-time graphical representation of API's



Secret Manager - Place to store and control access to private keys, passwords, certificates

# *Design Center: (Designing RAML)*

- Resource Name : Should be unique
- Method : Get, Post, Put, Delete, Fetch
- Request : Contains Body, Resource path, Uri path, Query param
- Response
- Http status: 200, 201, 202, 400, 401, 404, 500...
- Design center to Exchange
- Postman Testing
- Resource path, Query & Uri param

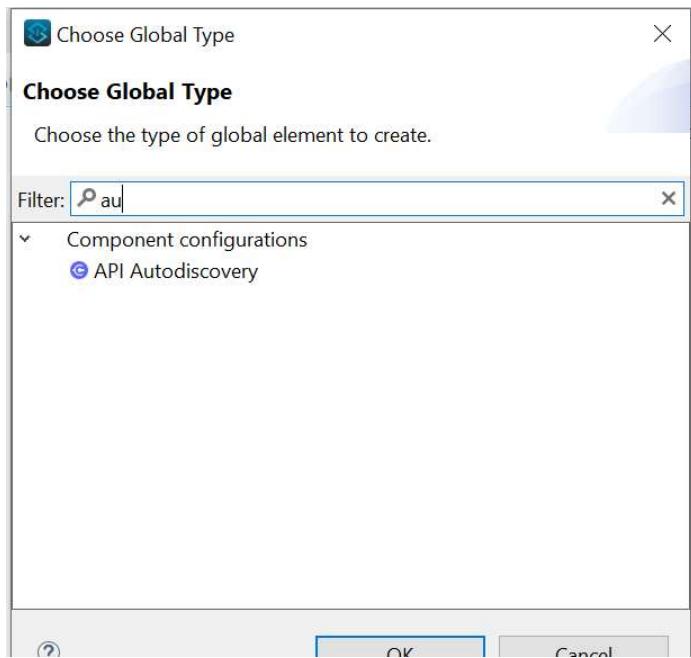
# API Manager

- Used to apply policies
- With Autodiscovery we cannot test the application with dummy ClientID & clientSecret
- What is Autodiscovery
  - API Auto Discovery is used to connect Runtime manager deployed applications with API created on the platform

The screenshot shows the API Manager interface for the 'government-sapi' API version v1. The top navigation bar includes 'doogle', a search icon, and user profile icons. The main page displays the API details: 'Sandbox' tab selected, 'government-sapi' API, version v1, 'Unregistered' status, Asset Version 1.0.0, Latest, Type RAML/OAS. Below this, there are sections for 'Add consumer endpoint', 'API Instance' (ID: 18183173), 'Label' (Add a label), and 'Tags' (ADD A TAG). A red circle highlights the 'Autodiscovery' section, which contains the text 'Autodiscovery' with a help icon, and 'API ID: 18183173'. A note at the bottom states: 'To complete the registration process, you need to connect this API to your Mule application using Autodiscovery. [Learn more](#)'.

# Adding AutoDiscoveryID in Studio

- AutodiscoveryID is used to interact between Runtime manager and API manager
- Copy AutodiscoveryID (Eg: 18183173)
- Apply AutodiscoveryID in Studio inside Global elements -> Create -> Search -> API Discovery -> API ID, FlowName (Always main flow)



# Policies

- Client\_id enforcement
- Rate limiting
- IP blocklist
- IP whitelist
- Jwt validation
- Basic Auth

The screenshot shows the API Manager interface for managing policies. A red circle highlights the 'Policies' link in the top-left corner of the main content area. The page displays a search bar and a 'Browse by category' section with tabs for ALL CATEGORIES (19), SECURITY (10), QUALITY OF SERVICE (4), TRANSFORMATION (2), COMPLIANCE (2), and TROUBLESHOOTING (1). The SECURITY tab is selected. Below the tabs, there are two sections: 'Security' and 'Quality of service'. The 'Security' section contains eight policy cards:

- Basic Authentication - LDAP (Status: ✓)
- OAuth 2.0 access token enforcement using Mule OAuth provider (Status: ✓)
- JWT Validation (Status: ✓)
- Detokenization (Status: You need permissions to apply this policy. Learn more) (Status: ⓘ)
- Basic Authentication - Simple (Status: ✓)
- XML threat protection (Status: ✓)
- IP Allowlist (Status: ✓)
- IP Blocklist (Status: ✓)

The 'Quality of service' section is partially visible at the bottom.

# Most Used Policy : Client ID Enforcement

The screenshot shows the 'API Administration (Sandbox)' interface with the 'government-sapi (v1) - Policies' tab selected. On the left, there's a sidebar with 'Sandbox' and a back-link to 'Policies'. The main area is titled 'Compliance' and contains two policy cards: 'Client ID Enforcement' and 'Cross-Origin Resource Sharing'. Below this is a 'Troubleshooting' section with a single card for 'Message Logging'.

API Administration (Sandbox) government-sapi (v1) - Policies

SANDBOX

← Policies

Compliance

**Client ID Enforcement**  
All calls to the API must include a client ID and client secret for an application that is registered...  
[Learn more](#)

**Cross-Origin Resource Sharing**  
CORS (Cross-Origin Resource Sharing) is a standard mechanism that allows JavaScript...  
[Learn more](#)

Troubleshooting

**Message Logging**  
Logs custom messages between policies and flow. Warning: If the payload is used as part of...  
[Learn more](#)

# Applying client enforcement policy

API Administration (Sandbox) government-sapi (v1) - Policies

SANDBOX

← Policies APIs / government-sapi / Policies / Configure Client ID Enforcement policy

Credentials origin

Origin of the Client ID and Client Secret credentials.

HTTP Basic Authentication Header

Custom Expression

Client ID Expression

DataWeave Expression to be used to extract the Client ID from API requests

```
##[attributes.headers['client_id']]
```

Client Secret Expression (Optional)

DataWeave Expression to be used to extract the Client Secret from API requests

```
##[attributes.headers['client_secret']]
```

Advanced options >

Configure policy duration, methods and resources

Add API ▾ (i) Filter by ▾  Search by API name, version or label

Status	API Name	Runtime	Label	Version	Instance	Error Rate	Total Requests
Unregistered	government-sapi	Mule 4	-	v1	18183173	No data	No data

Runtime Manager

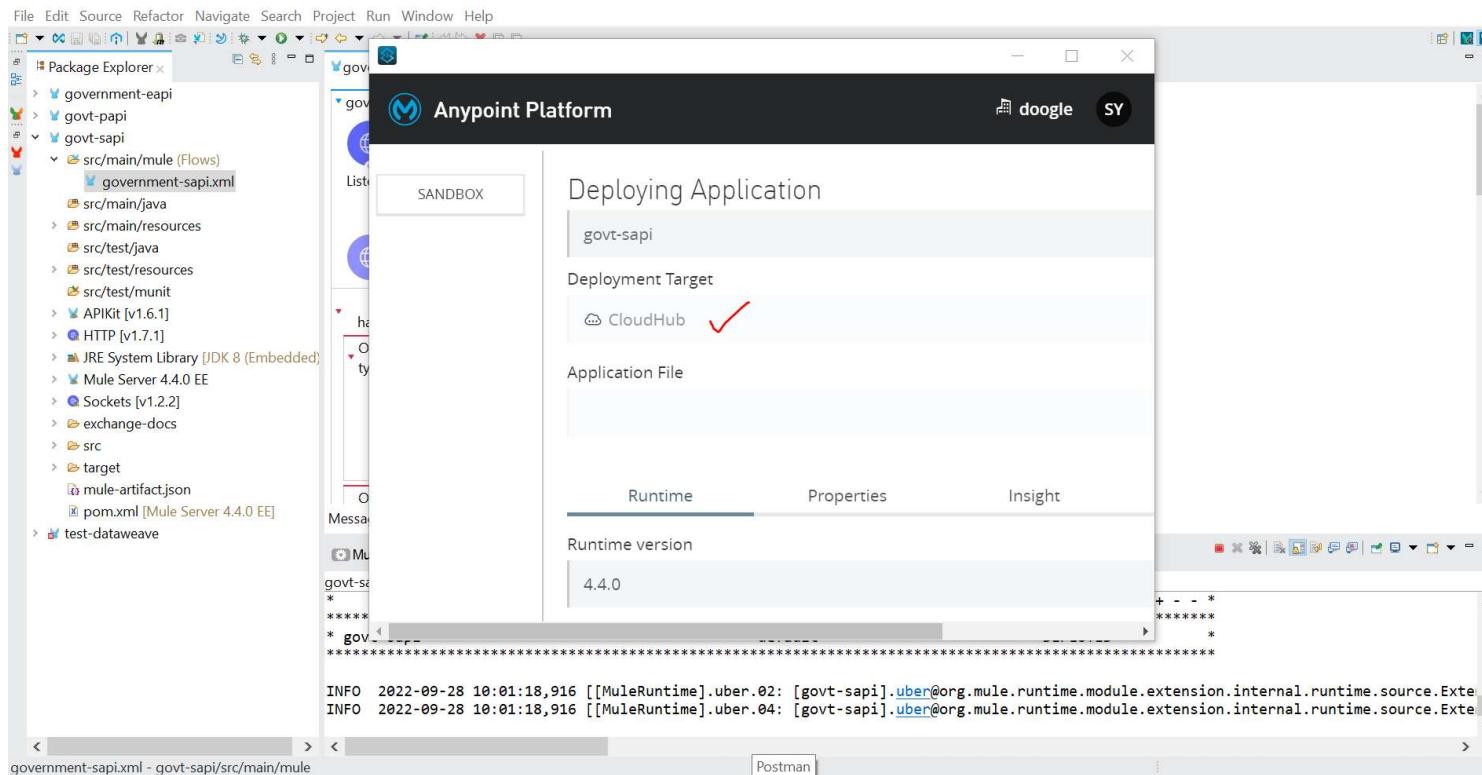
Deploying govt-papi to CloudHub

You may close this window at any time.

Open in Browser Close Window

Unregistered to Registered status via  
cloudhub deployment

# Runtime Manager: Deploy to cloudbhub via Studio



# AccessManagement: Get ClientID & ClientSecret

The screenshot shows the Access Management interface with the following details:

- Header:** doogle, SY
- Left Sidebar (ACCESS MANAGEMENT):**
  - Organization
  - Users
  - Roles
  - Environments** (selected)
  - Multi-Factor Auth
  - Identity Providers
  - Client Providers
  - Audit Logs
  - Connected Apps
  - External Access
- Left Sidebar (SETTINGS):**
  - Runtime Manager
  - Flow Designer
- Left Sidebar (SUBSCRIPTION):**
  - Runtime Manager
  - Object Store
- Bottom Left:** Try New Features
- Main Content:** Environments list (Design, Sandbox) with an Add environment button.
- Modal Dialog (Edit environment):**
  - Name:** Sandbox
  - Client ID:** af6b179a846844e4b5747ba3ed0d990b
  - Client Secret:** 42e2A73A43A94925b1aEC7514D3e0f3b (with a Hide button)
  - Buttons:** Delete Environment, Cancel, Update
- Bottom:** Postman

# Pasting ClientID & ClientSecret in Runtime Properties

The screenshot shows the Mule Runtime Manager interface for the application "govt-papi". The "Properties" tab is highlighted with a red oval. The properties listed are:

key	value
anypoint.platform.client_id	af6b179a846844e4b5747ba3ed0d990b
anypoint.platform.client_secret	42e2A73A43A94925b1aEC7514D3e0f3b
key	value

At the bottom right is a blue "Apply Changes" button.

## Policies & Uses

- Basic Auth -> Asks Username & Password
- ClientID Enforcement -> Requires `client_id` & `client_secret`
- Rate Limiting -> Send the set amount of request for particular time period
- IP Whitelist : Having Access to the Ip's which are in the list
- IP Blacklist: Does not have access to the IP's mentioned in the list

# Additional about RAML & Status codes:



Creating folder for code simplicity (! include  
FILENAME in Type -- application/json)

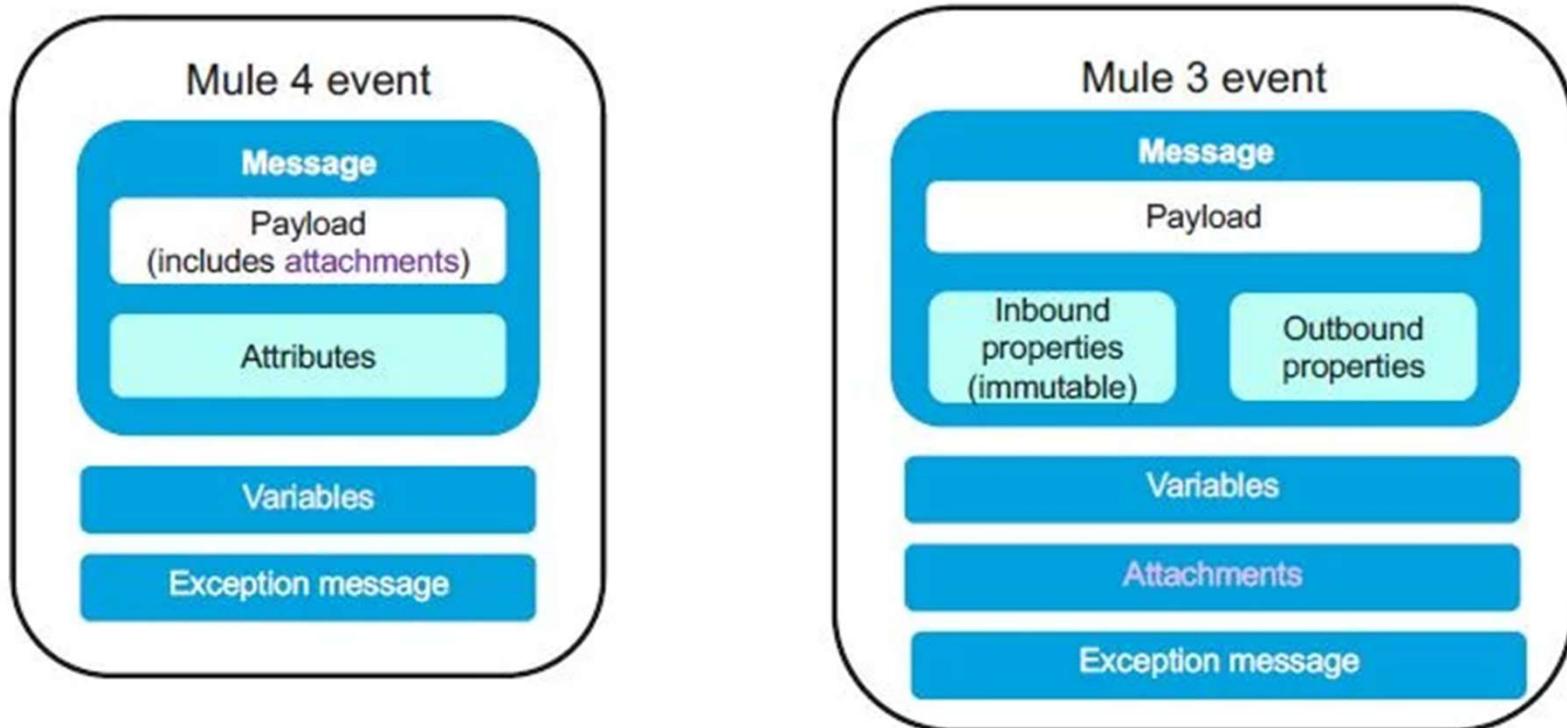


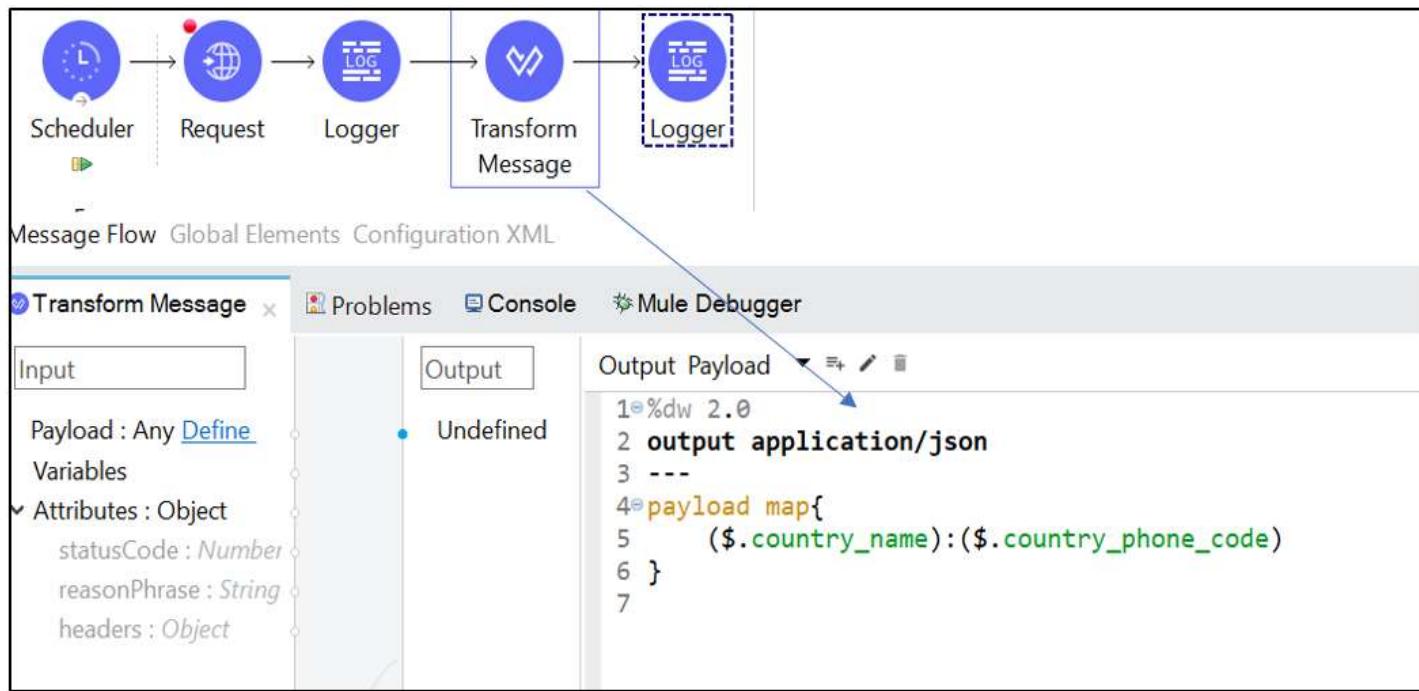
<https://http.cat/>  
(Example status codes)



<https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/>  
(More about RAML)

## Mule3 vs Mule4 Structure





# What is Payload? (Part of Mule Message)

Payload is an actual message content. Payload can be overridden

# What is Attribute? (Part of Mule Message)

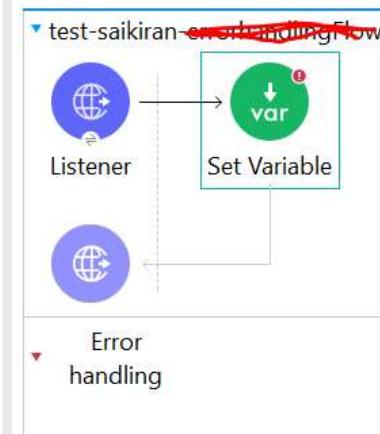
This provides Metadata such as queryparams, File size etc. These are immutable(Cannot be changed)

The screenshot shows the Anypoint Studio interface with the 'Output' tab selected. A search bar at the top contains the placeholder 'type filter text'. Below it, a tree view of message components. The 'Attributes' section is expanded, showing the following properties:

- clientCertificate : Object?
- headers : Object?
- listenerPath : String?
- method : String?
- queryParams : Object?
- queryString : String?
- relativePath : String?
- remoteAddress : String?
- requestPath : String?
- requestUri : String?
- scheme : String?
- uriParams : Object?
- version : String?

# What are Variables? (Part of Mule event)

- These are which you can store some kind of data or information and can be used accross the application as long as you connect the flow with flow-ref
- Set Payload:
  - Used to Set the variable
  - Set variable can be called in Payload
  - Example: "My Name is " ++ vars.name ++ "My id is " ++vars.id



Message Flow Global Elements Configuration XML

Set Variable Console Problems Mule

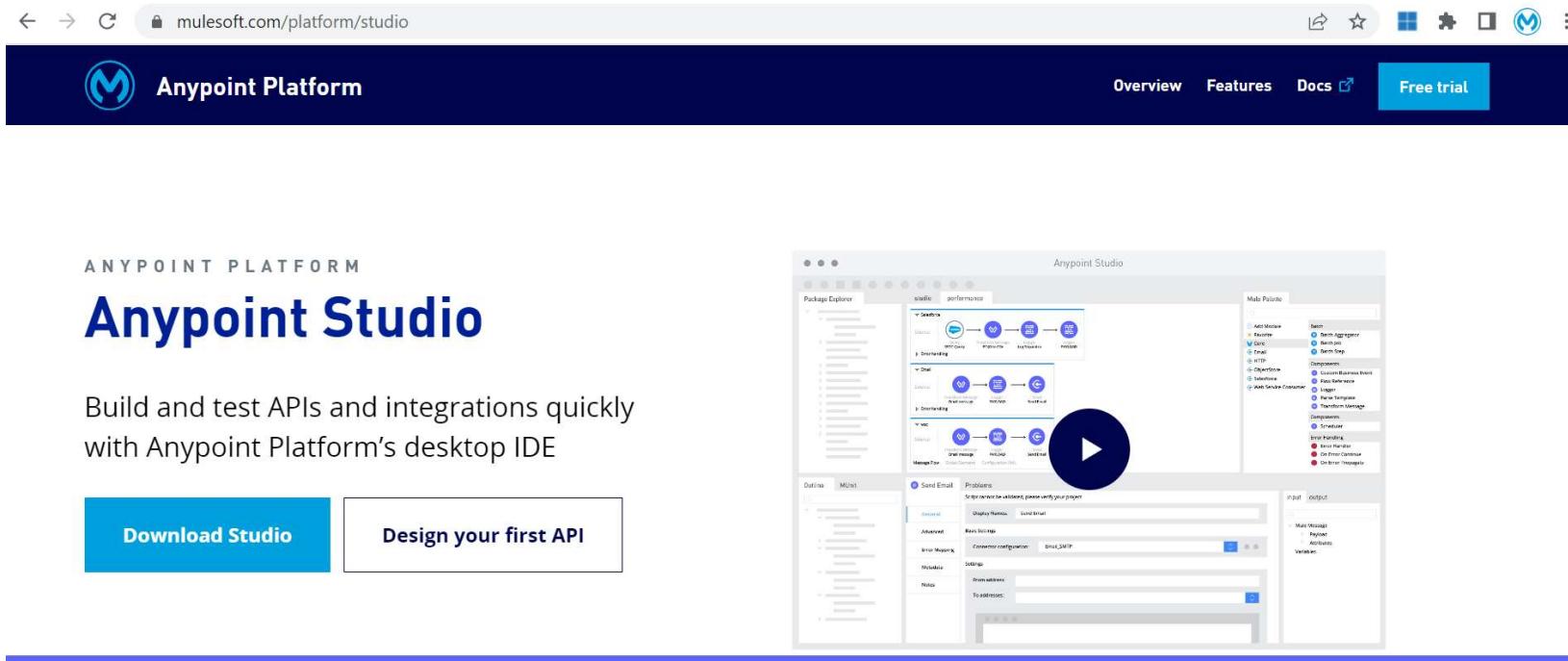
General

There are no errors.

MIME Type	Display Name: Set Variable
Metadata	Settings
Notes	Name: id
Help	Value: <input type="text" value="100"/> <input type="button" value="fx"/>

# Anypoint Studio

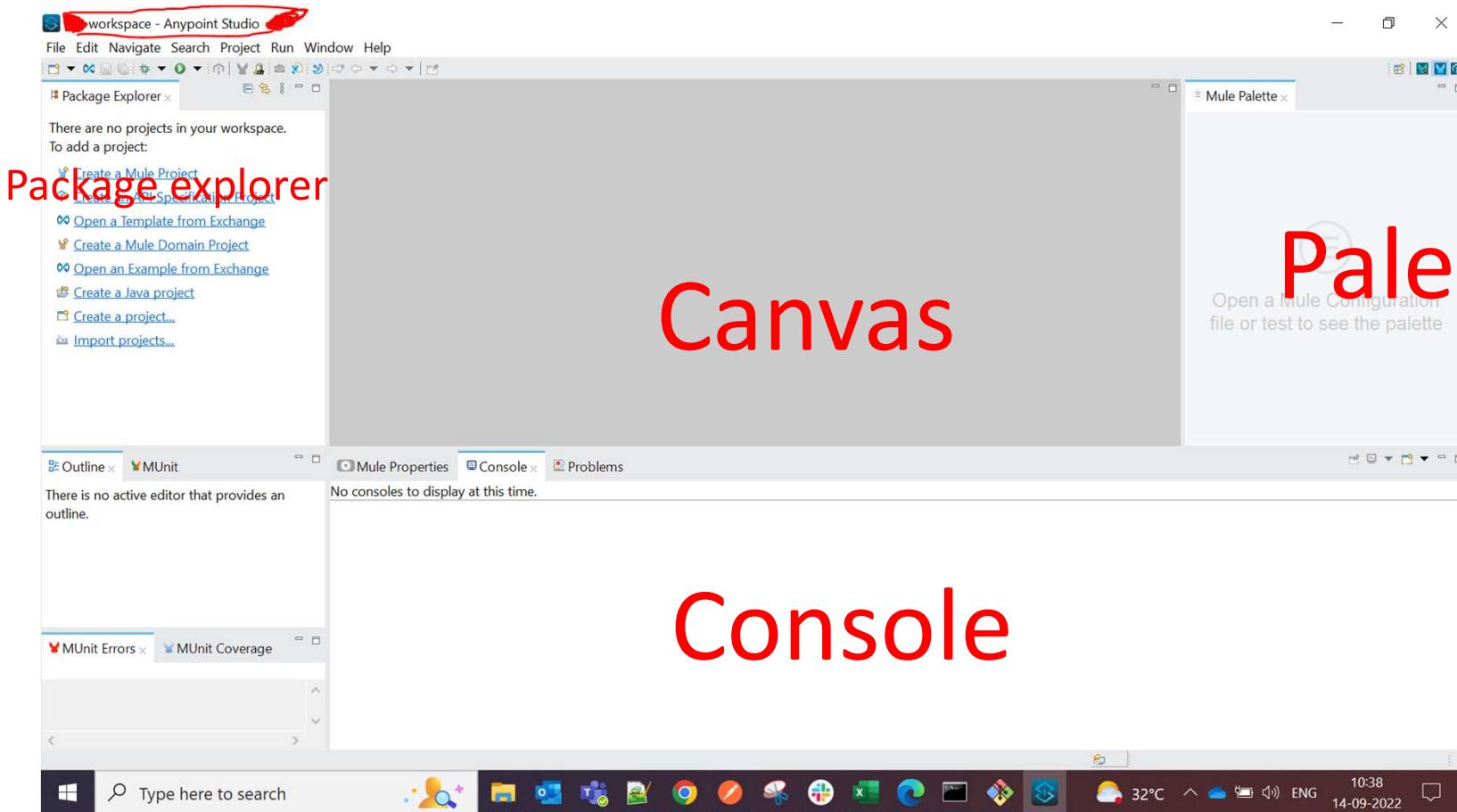
- Link to Install:  
<https://www.mulesoft.com/platform/studio>



The screenshot shows the Anypoint Platform website with the following elements:

- Header:** mulesoft.com/platform/studio
- Logo:** Anypoint Platform logo
- Navigation:** Overview, Features, Docs, Free trial
- Main Content:** A large image of the Anypoint Studio IDE interface.
- Studio Interface Details:**
  - Package Explorer:** Shows a project structure with nodes like "HelloWorld", "Ordering", and "Order".
  - Toolbars:** "Send", "Performance", "Mule Palette", "Data", "Misc", "Send Email", "Problems", "Advanced", "Error Mapping", "Metadata", "Notes".
  - Mule Palette:** A list of Mule components including "Add Message", "Flow", "Get", "HTTP", "JDBC", "JMS", "Script", "Select", "Set", "Text", "Transform", "XPath", "File", "Batch Aggregator", "Batch Stop", "Logger", "Parallelepiped", "On Error Continue", and "On Error Throttle".
  - Input/Output:** Options for "Mail Message", "Payload", "Attachments", and "Variables".
- Bottom Buttons:** Download Studio, Design your first API

# Anypoint Studio Explanation



Canvas

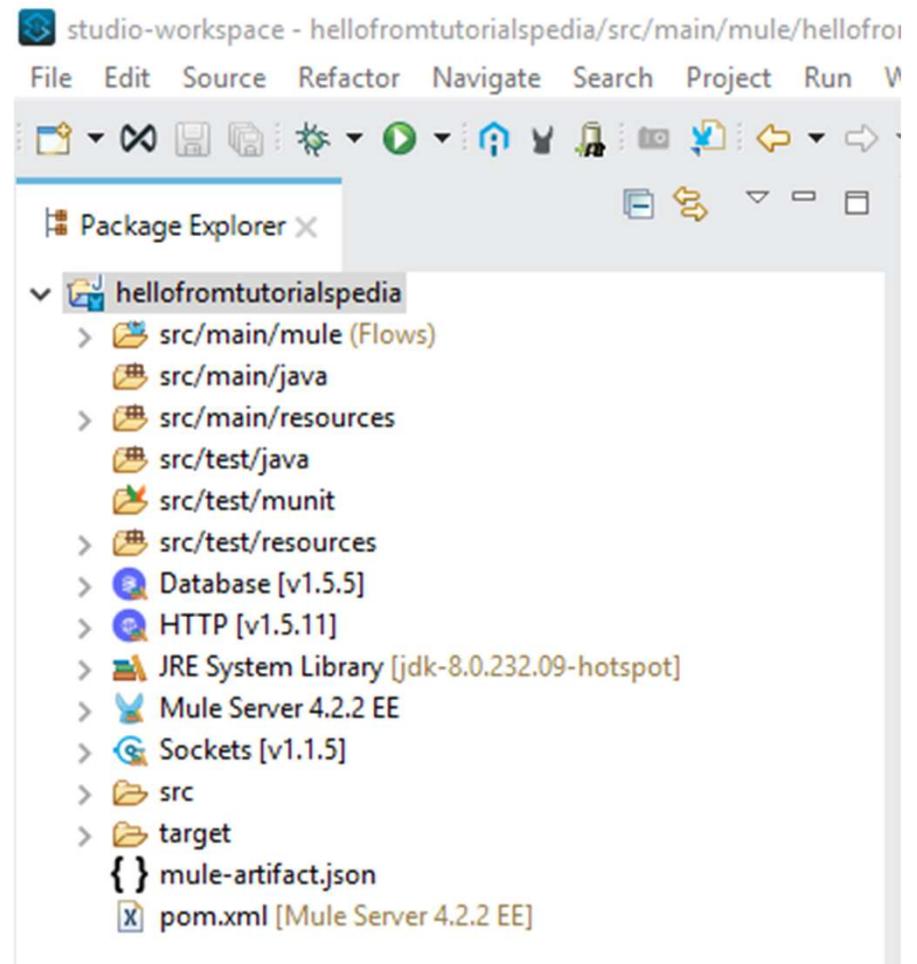
Palette

Console

# Important points about Mule

---

- Mulesoft is a Mavenized project
- All Mule xml's are placed under src/main/mule
- Other files can be placed under: src/main/resources or src/test/resources
- Mule-artifact.json
- Each mule app xml has – Message flow(graphical view), Global elements(all config details), Configuration xml (xml version or graphical view)



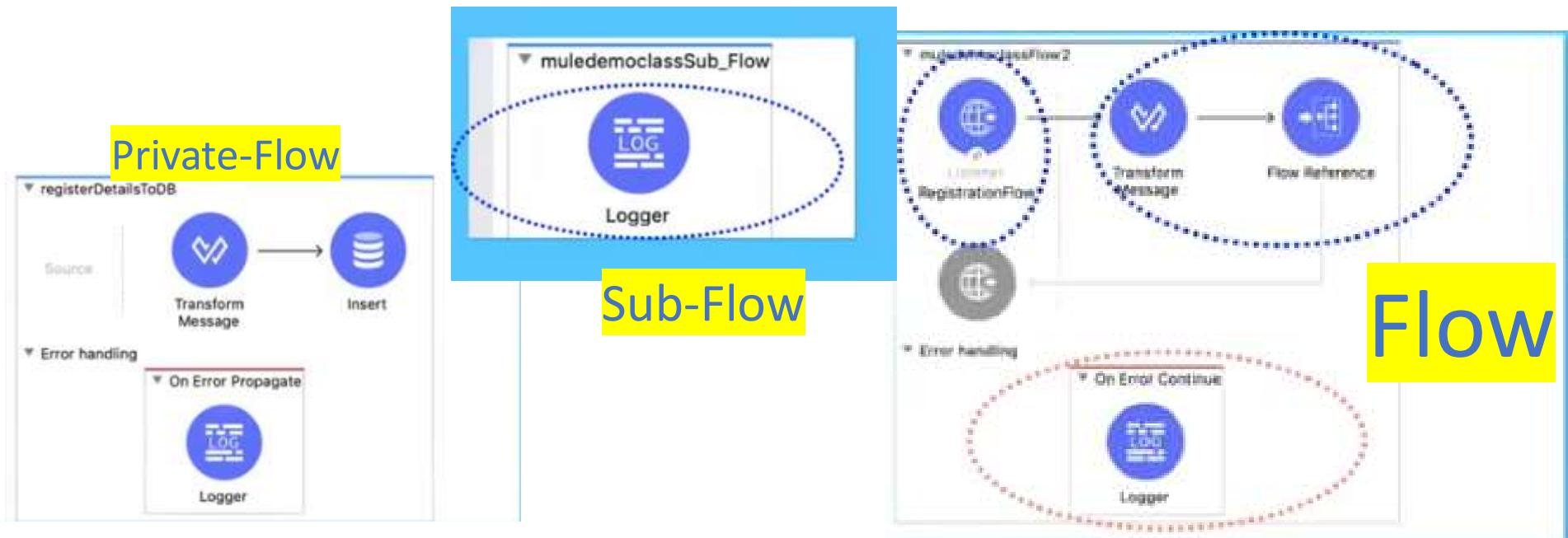
# Flow, Sub-flow & Private flow

**Flow** is a message processing block that has its own processing strategy and exception handling strategy. Used in integration tasks, data processing, connecting applications, event processing, etc.

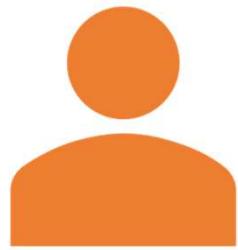
**Subflow** always processes messages synchronously but inherits processing strategy and exception handling strategy from the calling flow. It can be used to split common logic and be reused by other flows.

**Private flow** does not have source define. It can be synchronous or asynchronous based on the processing strategy selected. Also they have their own exception handling strategy. Allows you to define different threading profile.

Flow	Sub Flow	Private Flow
Flow have <b>Source, Process and Error handling</b> part	A flow that do not have <b>Source and Error handling</b>	A flow that have <b>Source, Process &amp; Error handling</b>
We will give reference to other flows based on requirement	Having only Process part	We will call it as private flow when source part is empty



# Automated vs Manual Project

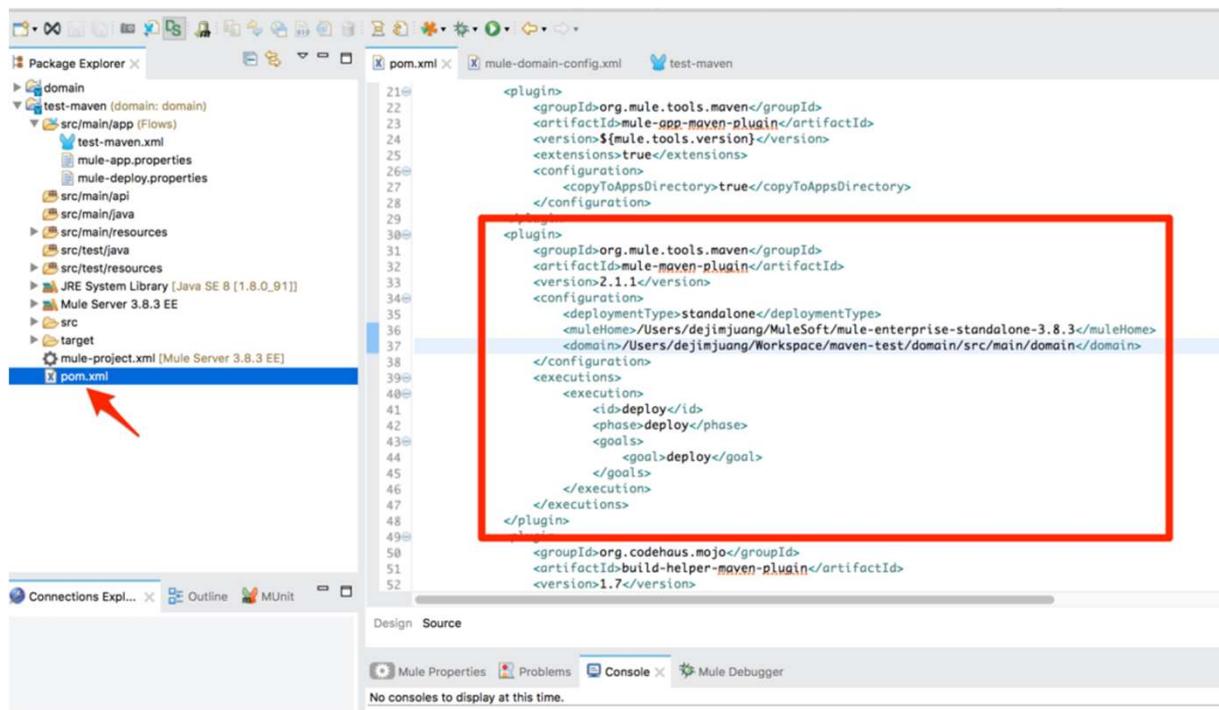


Automated : Scheduler + Logger



Manual: Listener+logger

# Pom.xml

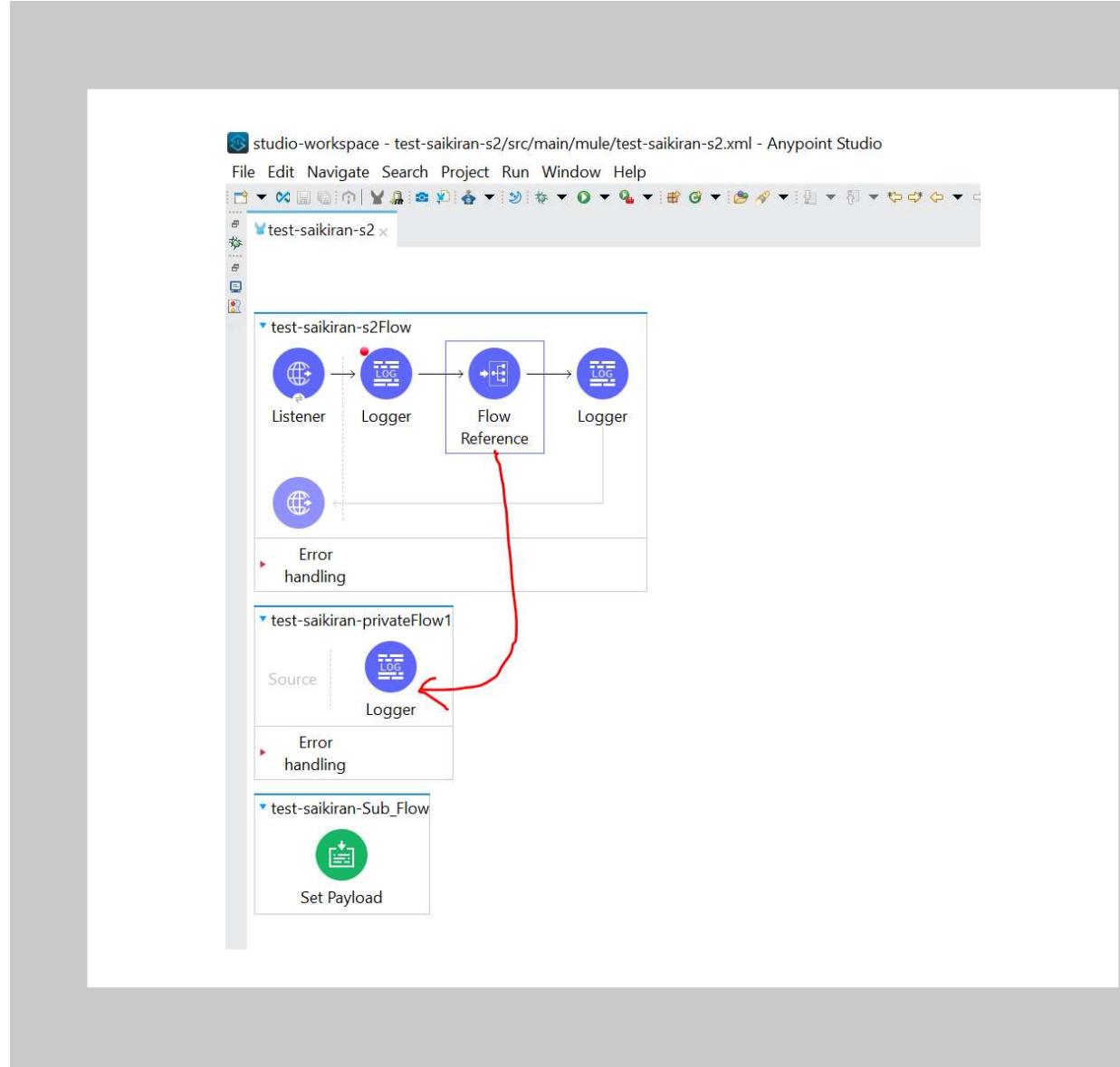


```
<plugin>
    <groupId>org.mule.tools.maven</groupId>
    <artifactId>mule-app-maven-plugin</artifactId>
    <version>$\{mule.tools.version\}</version>
    <extensions>true</extensions>
    <configuration>
        <copyToAppsDirectory>true</copyToAppsDirectory>
    </configuration>
</plugin>
<plugin>
    <groupId>org.mule.tools.maven</groupId>
    <artifactId>mule-maven-plugin</artifactId>
    <version>2.1.1</version>
    <configuration>
        <deploymentType>standalone</deploymentType>
        <muleHome>/Users/dejimjuang/MuleSoft/mule-enterprise-standalone-3.8.3</muleHome>
        <domain>/Users/dejimjuang/Workspace/maven-test/domain/src/main/domain</domain>
    </configuration>
    <executions>
        <execution>
            <id>deploy</id>
            <phase>deploy</phase>
            <goals>
                <goal>deploy</goal>
            </goals>
        </execution>
    </executions>
</plugin>
<groupId>org.codehaus.mojo</groupId>
<artifactId>build-helper-maven-plugin</artifactId>
<version>1.7</version>
```

- The pom.xml file contains the core information about a project and its configuration details including its dependencies, build directory, source directory, test source directory, plugin, goals etc. Maven reads the pom.xml file, then executes the goal.

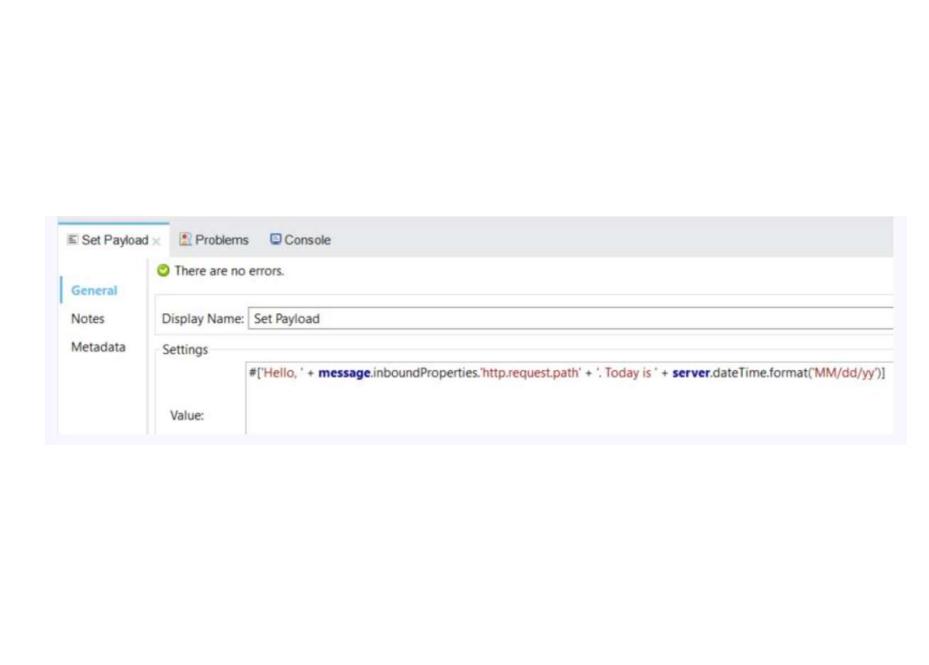
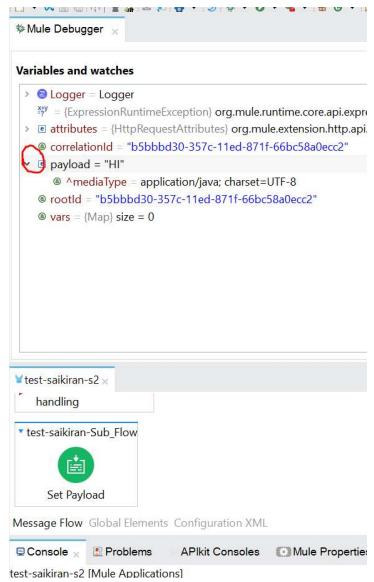
# Flow reference

- Flow Reference breaks the Mule application into discrete and reusable units
- Flow Reference component in a flow to direct Mule to send a message to another flow for processing



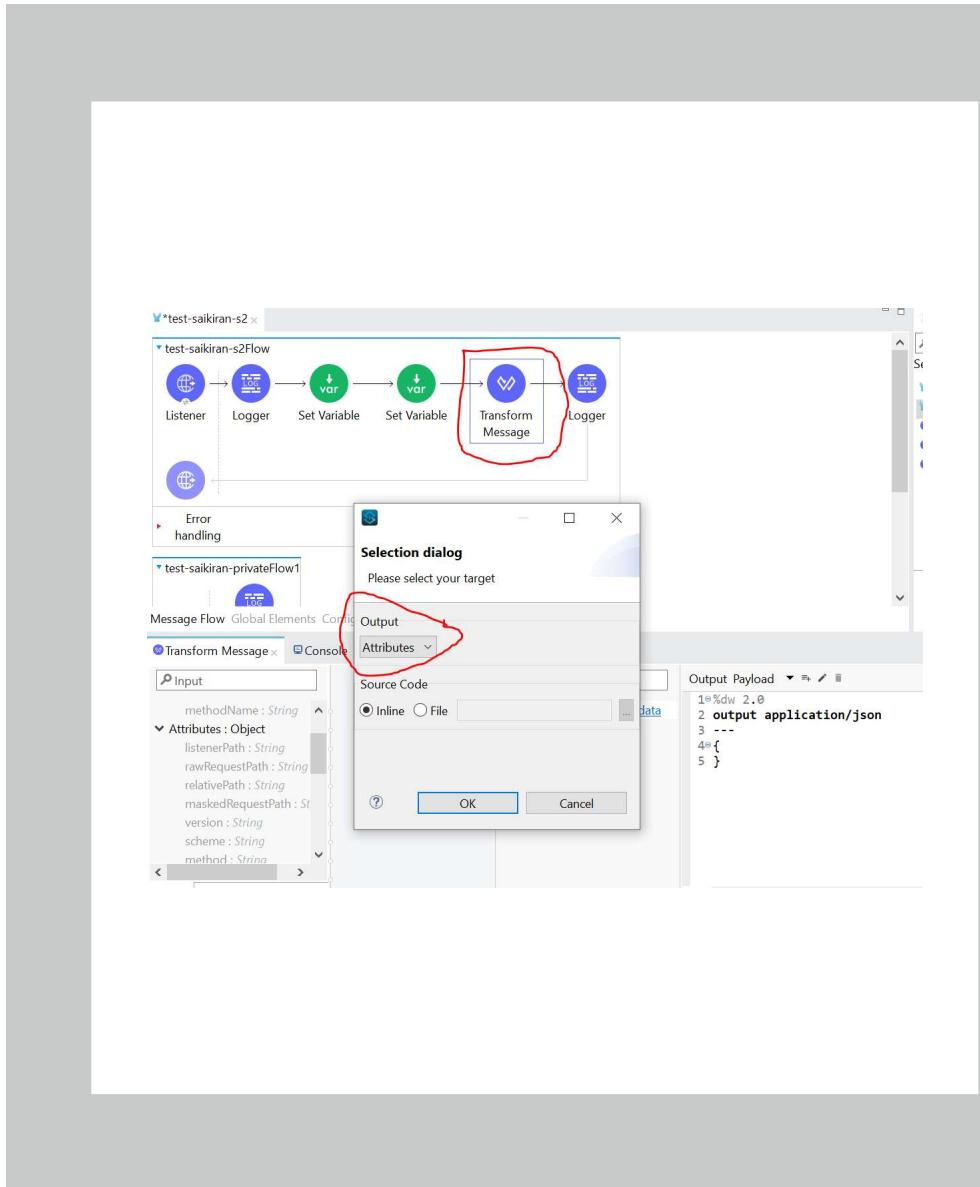
# What is Set Payload

- This transformer sets the payload to whatever you define. The payload can be a literal string or a Mule Expression.



# Transform Message

- It also works same as Payload but it will also do the work of set variable
- You can add n number of set variables to Transform message but only 1 payload can be added

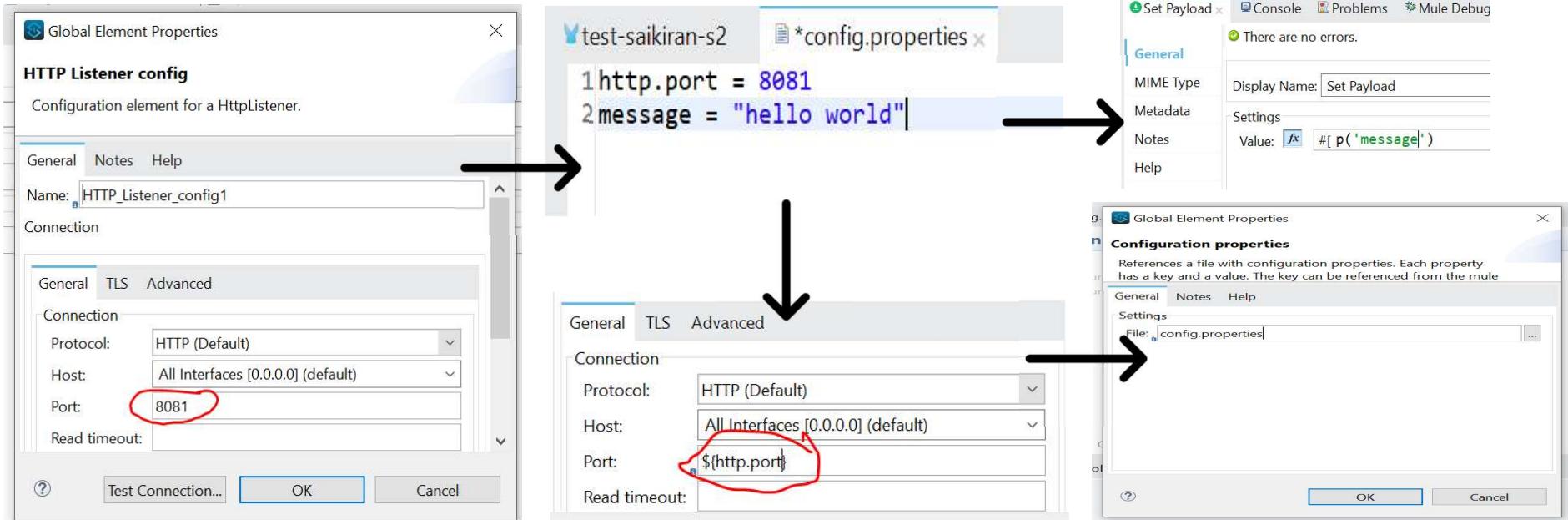


# Configuration properties

- Properties or values that are used for best practice to extract some values from “property” files using corresponding key to avoid hard coding values
- We can extract property values from property file by 2 ways
  - \${} :- Used at anywhere (Config/Dataweave)
  - p('') :- Used in Dataweave expression language scope

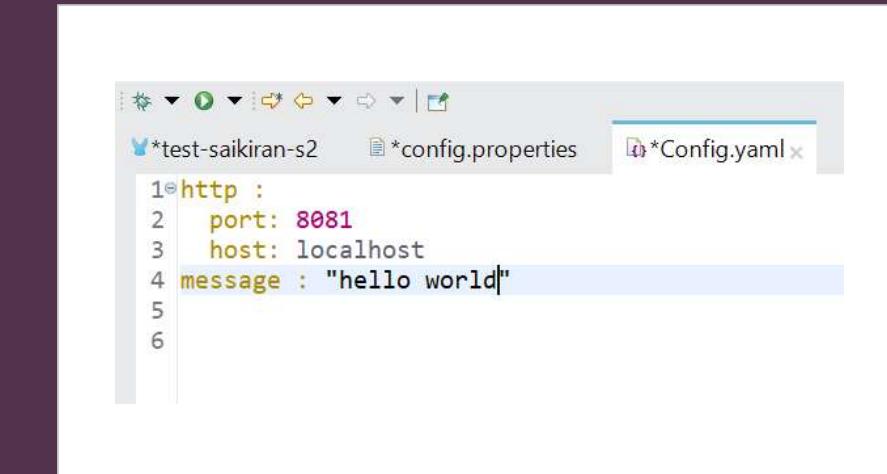
## How to create config

- Navigate to project -> src/main/resources -> RightClick -> File -> Create File -> config.properties (File Name)
- After creating the file -> Go to Global elements -> Create -> configuration properties -> Name the file name which you created (config.properties)
- P(): Can be used in Payload (eg: p('message'))



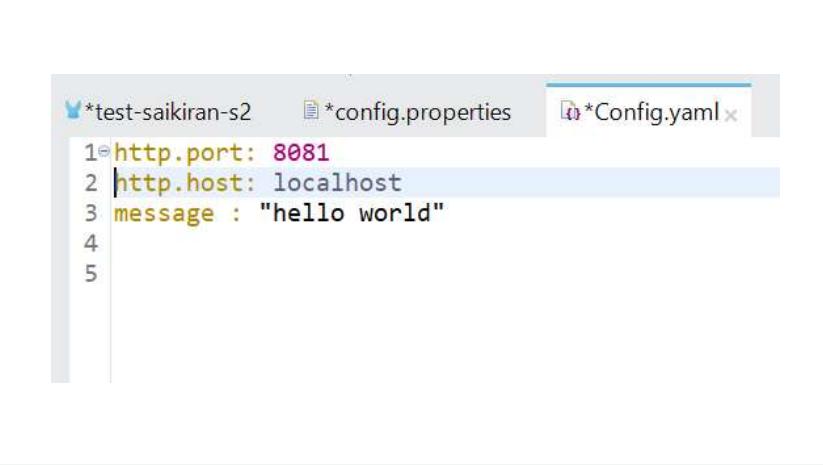
# YAML

- YAML is a human-readable data-serialization language. It is commonly used for configuration files and in applications where data is being stored or transmitted
- Yaml structure (http.port: 8081) where config.properties are like (http.port = 8081)



The screenshot shows a code editor window with three tabs at the top: \*test-saikiran-s2, \*config.properties, and \*Config.yaml. The \*Config.yaml tab is active. The code in the editor is:

```
1 http :  
2   port: 8081  
3   host: localhost  
4 message : "hello world"  
5  
6
```



The screenshot shows a code editor window with three tabs at the top: \*test-saikiran-s2, \*config.properties, and \*Config.yaml. The \*Config.yaml tab is active. The code in the editor is:

```
1 http.port: 8081  
2 http.host: localhost  
3 message : "hello world"  
4  
5
```

# DataWeave - Introduction

---

- DataWeave is a MuleSoft expression language for accessing and transforming data
- We can also access specific field inside payload
- We can extract all kind of Data and Transform it through DataWeave
- Transfer of data from one Format to another format is possible through DataWeave
  - Eg: XML to JSON, CSV to JSON etc...

More about DataWeave:

[https://www.tutorialspoint.com/mulesoft/mulesoft\\_dataweave\\_language.htm](https://www.tutorialspoint.com/mulesoft/mulesoft_dataweave_language.htm)

# DataWeave :- Preview Feature

Through preview we can see how the output generates without execution

It will help us to check whether if there are any syntax errors before execution

It will help us reduce time for debugging and development

# Preview Through Sample Data

Output Payload

```
1 dw 2.0
2 var a ={
3   "name": "Saikiran",
4   "id": 101,
5   "org": "cognizant"
6 }
7 output application/json
8 ---
9   "Hi " ++ a.name
```

Tree Text

Preview

The preview interface shows a sample payload on the left and a flow diagram on the right.

**Sample Payload:**

```
"Hi Saikiran"
```

**Flow Diagram:**

```
graph LR; Listener((Listener)) --> Transform[Transform Message]; Transform --> ErrorHandling[Error handling]
```

The flow starts with a Listener node, followed by a Transform Message node, and finally an Error handling node.

# Preview through Payload data

- Run the application
- Set the payload data in Transform message
- Request the data in postman to verify in response

The image shows two side-by-side screenshots. On the left is the Mule Debugger interface, specifically the 'Output Payload' tab. It displays a Mule expression (dw 2.0) that concatenates a greeting, the payload's name, its ID, and its organization. On the right is the Postman application, showing a GET request to 'http://localhost:8081/data'. The request body is a JSON object with fields 'name', 'id', and 'org'. The response body shows the concatenated string from the Mule expression.

**Mule Debugger (Left):**

```
1%dw 2.0
2 output application/json
3 ---
4|   "Hi " ++ payload.name ++ "Your id is"
5   ++ payload.id ++ "Org is" ++ payload.org
```

**Postman (Right):**

GET http://localhost:8081/data

Body (Pretty)

```
1
2   ...
3     "name": "Saikiran",
4     "id": 101,
5     "org": "cognizant"
```

Body (Raw)

```
1 "Hi SaikiranYour id is101Org iscognizant"
```

# Object & Array

```
[101, "Saikiran", "Cognizant"]
```

```
{  
    "name": "vasavi",  
    "id": 101,  
    "org": "IBM"  
    "Location": "Hyderabad"  
}
```

```
{  
    "name": "vasavi",  
    "id": 101,  
    "org": "IBM",  
    "Location":  
        {  
            "Street": "DLF Road",  
            "City": "Hyderabad",  
            "State": "Telangana"  
        }  
}
```

```
{  
    "name": "vasavi",  
    "id": 101,  
    "org": "IBM",  
    "Location":  
        [  
            "DLF Road", "Hyderabad", "Telangana"  
        ]  
}
```

- Object is denoted with in {}, and it is having key-value pair
- Array is denoted with [ ], It have only values
- Link to Validate JSON:
  - <http://jsonviewer.stack.hu/>

# Transforming One data type to Another

- In Mule3, We have different connectors to convert data from one type to another
- In Mule4, we use DataWeave to achieve this
  - Examples
    - JSON to XML
    - XML to CSV
    - JSON to CSV
    - XML to JSON



The image displays four separate windows from a transformation tool, each showing a code editor and a preview pane. The windows are labeled with their respective output formats: JSON, CSV, Java, and XML.

- JSON:** Shows a JSON object with fields name, id, and org. The preview pane shows the JSON structure: {"name": "Saikiran", "id": "100", "org": "doogle"}. A red circle highlights the word "output" in the code editor.
- CSV:** Shows a CSV output with the same data. The preview pane shows the CSV row: name,id,org Saikiran,100,doogle. A red circle highlights the word "output" in the code editor.
- Java:** Shows a Java Object with fields name, id, and org. The preview pane shows the object structure: Object { name = "Saikiran", id = "100", org = "doogle" }. A red circle highlights the word "output" in the code editor.
- XML:** Shows an XML output with a Details element containing name, id, and org. The preview pane shows the XML structure: <?xml version='1.0' encoding='UTF-8'?><Details><name>Saikiran</name><id>100</id><org>doogle</org></Details>. A red circle highlights the word "output" in the code editor.

```

Output Payload | Preview
3 var a ={
4   "name": "Saikiran",
5   "id": "100",
6   "org": "doogle"
7 }
8 output application/json
9 ---
10
11 a
12

Output Payload | Preview
3 var a ={
4   "name": "Saikiran",
5   "id": "100",
6   "org": "doogle"
7 }
8 output application/csv
9 ---
10
11 a
12

Output Payload | Preview
3 var a ={
4   "name": "Saikiran",
5   "id": "100",
6   "org": "doogle"
7 }
8 output application/java
9 ---
10
11 a
12

Output Payload | Preview
3 var a ={
4   "name": "Saikiran",
5   "id": "100",
6   "org": "doogle"
7 }
8 output application/xml
9 ---
10
11 "Details":a
12

```

## Transformation Data Pictures

# Commonly Used operators

SOME MOST IMPORTANT AND COMMONLY USED OPERATORS			
Operator	Can be applied on input which is/are	Output type	Used for
map	On Arrays only	Array	Array of Object
mapObject	On Objects only	Object	Output is an Object
reduce	On Arrays	Anything	To reduce into given expression
pluck	On Objects	Array	Same as mapObject, only difference is the output is returned as array instead of Object
flatten	On arrays	Single set of Array	Turns into set of subarrays to single array

Function	Input	Output	Description
pluck	Object	Array	Which iterates over an object uses anonymous mapper function and returns the output as an array
mapObject	Object	Object	Extracts key/value pairs or indices from the object
flatten	Multiple Arrays	Single Array	Iterates over an object, extract the key/value pairs or indices and returns an object Which merge the array of arrays into single array

- Map & Map Object syntax:

```
map((value, index) ->{ })
```

```
mapObject((value, key, index)->{ })
```

map operators is used in Arrays

mapObject is used for Objects

map operators cannot be used inObjects

mapObject cannot be used for Arrays

\$: Gives the value

\$\$: Gives the index of the field

\$\$\$: Gives the Key of the field

Map working  
(\$ working)

Output Payload

```
1%dw 2.0
2
3@var a =[{"id":100,"name":"Saikiran","Org":"Cognizant"}, {"id":101,"name":"kiran","Org":"Cognizant"}, {"id":103,"name":"Sai","Org":"Cognizant"}]
4
5output application/json
6
7a map {
8    "candidateName": $.name,
9    "id": $.id,
10   "company": $.Org,
11   "UniqueId": 100 ++| $.id++"-"+$.name
12 }
```

Preview

```
[{"candidateName": "Saikiran", "id": 100, "company": "Cognizant", "UniqueId": "100100-Saikiran"}, {"candidateName": "kiran", "id": 101, "company": "Cognizant", "UniqueId": "100101-kiran"}, {"candidateName": "Sai", "id": 103, "company": "Cognizant", "UniqueId": "100103-Sai"}]
```

# Map with (value,Index)

- When you got map and give ctrl + space,  
It gives as below format
  - a map ((value, index) ->{})
  - Upper keyword is used for  
Org, please check

The screenshot shows a code editor interface with two panes. The left pane is titled "Output Payload" and contains the following Groovy script:

```
1%dw 2.0
2
3var a =[
4{
5  "id":100,
6  "name":"Saikiran",
7  "Org":"Cognizant"
8},
9{
10 "id":101,
11 "name":"kiran",
12 "Org":"Cognizant"
13},
14{
15 "id":103,
16 "name":"Sai",
17 "Org":"Cognizant"
18}
19]
20
21output application/json
22---
23
24a map ((item, index) ->{
25  "candidateName":upper(item.Org),
26  "id":item.id,
27  "company":item.Org,
28  "Dollar": index
29} )
30
```

The right pane is titled "Preview" and displays the resulting JSON array:

```
[{"candidateName": "COGNIZANT", "id": 100, "company": "Cognizant", "Dollar": 0}, {"candidateName": "COGNIZANT", "id": 101, "company": "Cognizant", "Dollar": 1}, {"candidateName": "COGNIZANT", "id": 103, "company": "Cognizant", "Dollar": 2}]
```

# MapObject : for Objects only

DataWeave Playground

Export Import Tutorial Playground

PAYLOAD	JSON	SCRIPT	OUTPUT
<pre>1  [{" 2      "id": 101, 3      "name":"Saikiran", 4      "location":"Vizag" 5    }, 6    { 7      "id": 102, 8      "name":"Pragna", 9      "location":null 10   }, 11   { 12     "id": 103, 13     "name":"Hero", 14     "location":"Pvp" 15   }, 16   { 17     "id": 104, 18     "name":"Ramu", 19     "location":null 20   }]</pre>		<pre>1  %dw 2.0 2  output application/json 3  --- 4  payload[2] mapObject 5  { 6    "Employee ID": \$ 7  } 8 9 10  </pre>	<pre>1  { 2    "Employee ID": 103, 3    "Employee ID": "Hero", 4    "Employee ID": "Pvp" 5  }</pre>

The screenshot shows the DataWeave Playground interface. The PAYLOAD tab displays a JSON array of four objects. The third object, which has an 'id' of 103 and a 'name' of 'Hero', is highlighted with a red oval. The SCRIPT tab contains a DataWeave script that uses the `mapObject` function on the third payload item. The resulting OUTPUT is a JSON object where the key 'Employee ID' maps to the value 103, the key 'Employee ID' maps to the string 'Hero', and the key 'Employee ID' maps to the string 'Pvp'. The entire output object is also highlighted with a red oval.

# mapObject((key, value, index)->{})

EXPERIENCE INNOVATION, UNLEASHED. WATCH THE HIGHLIGHTS FROM CONNECT '22

DataWeave Playground

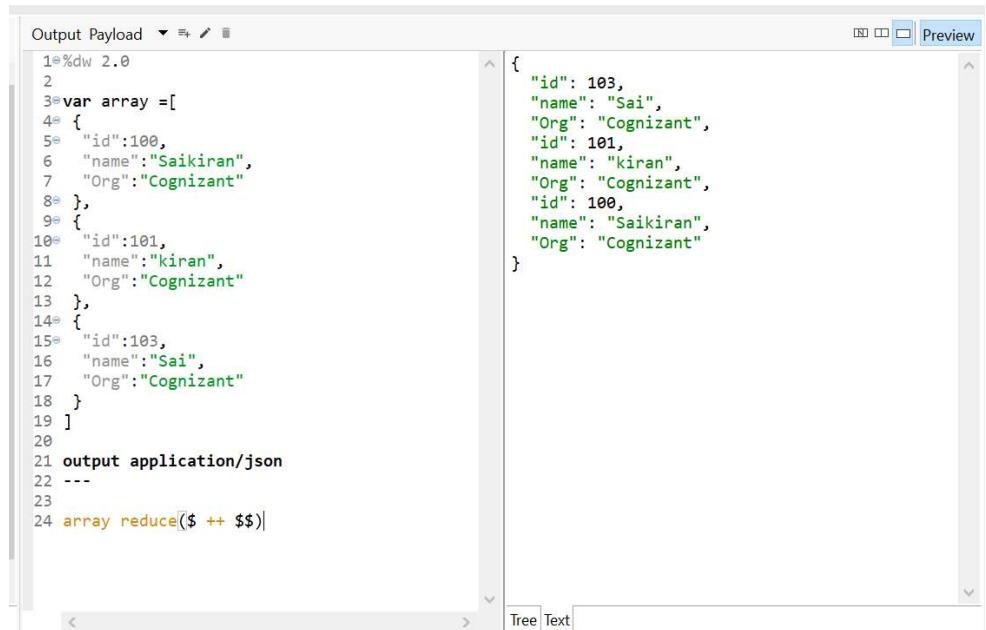
Export Import Tutorial Playground

PAYLOAD	JSON	SCRIPT	OUTPUT	JSON
<pre>1  [{ 2    "id": 101, 3    "name":"Saikiran", 4    "location":"Vizag" 5  }, 6  { 7    "id": 102, 8    "name":"Pragna", 9    "location":null 10 }, 11 { 12   "id": 103, 13   "name":"Hero", 14   "location":"Pvp" 15 }, 16 { 17   "id": 104, 18   "name":"Ramu", 19   "location":null 20 }]</pre>		<pre>1  %dw 2.0 2  output application/json 3  --- 4  payload[2] mapObject ((value, key, index) -&gt; 5  { 6    "Key":key, 7    "Value":value, 8    "Index":index 9  } ) 10 11 12</pre>	<pre>1  { 2    "Key": "id", 3    "Value": 103, 4    "Index": 0, 5    "Key": "name", 6    "Value": "Hero", 7    "Index": 1, 8    "Key": "location", 9    "Value": "Pvp", 10   "Index": 2 11 }</pre>	

LOG VIEWER API REFERENCE ©2022 MuleSoft LLC, a Salesforce company

# Operator: Reduce

- Reduce function is used to reduce Array into a single object
- Implemented only on arrays



The screenshot shows a software interface with a "Output Payload" panel. On the left, there is a code editor window containing the following Groovy script:

```
1@%dw 2.0
2
3@var array =[{
4@ {
5@   "id":100,
6@   "name":"Saikiran",
7@   "Org":"Cognizant"
8@ },
9@ {
10@   "id":101,
11@   "name":"kiran",
12@   "Org":"Cognizant"
13@ },
14@ {
15@   "id":103,
16@   "name":"Sai",
17@   "Org":"Cognizant"
18@ }
19 ]
20
21 output application/json
22 ---
23
24 array reduce($ ++ $$)|
```

On the right, there is a "Preview" window showing the resulting JSON object:

```
{
  "id": 103,
  "name": "Sai",
  "Org": "Cognizant",
  "id": 101,
  "name": "kiran",
  "Org": "Cognizant",
  "id": 100,
  "name": "Saikiran",
  "Org": "Cognizant"
}
```



# demo.txt

# Reduce: Array of Objects

EXPERIENCE INNOVATION, UNLEASHED. WATCH THE HIGHLIGHTS FROM CONNECT '22

DataWeave Playground

PAYOUT	JSON	SCRIPT	OUTPUT
<pre>1 { 2   "Orders": [ 3     [ 4       { 5         "id": "10121", 6         "Customer": { 7           "name": "Saikiran", 8           "phone": "9490541503", 9           "email": "test@gmail.com" 10          }, 11        "order": { 12          "purchaseitems": [ 13            [ 14              { 15                "item": "iphone11", 16                "rate": 43000, 17                "Qty": 2 18              }, 19              { 20                "item": "poco", 21                "rate": 18000, 22                "Qty": 1 23              }, 24              { 25                "item": "MI", 26                "rate": 10000, 27                "Qty": 1 28              } 29            ] 30          } 31        } 32      ] 33    } 34 }</pre>	<pre>1 %dw 2.0 2 3 output application/json 4 --- 5 payload.Orders map( 6   "OrderID": \$.id, 7   "customerName": \$.Customer.name, 8   "customerNo": \$.Customer.phone, 9   "Items": \$.order.purchaseitems.item reduce (\$ ++ ", "++ \$\$ ), 10  "Quantity": \$.order.purchaseitems.Qty reduce (\$+\$\$ ), 11  "totalCost": \$.order.purchaseitems.rate reduce(\$+\$\$) 12 )</pre>	<pre>1 [ 2   { 3     "OrderID": 10121, 4     "customerName": "Saikiran", 5     "customerNo": "9490541503", 6     "Items": "MI, poco, iphone11", 7     "Quantity": 4, 8     "totalCost": 71000 9   } 10 ]</pre>	

Export Import Tutorial Playground

# Operator: Pluck

- Based on customer needs it plucks the value (it will be in the format of Array of Objects)

The screenshot shows a development environment interface. On the left, there is a code editor window titled "Output Payload" containing Groovy script. The script defines a variable `p` as a list of three objects, each with a key `"The value is"` and a corresponding value ("100", "Saikiran", or "Cognizant"). It then outputs the list as JSON. On the right, there is a "Preview" window showing the resulting JSON output.

```
dw 2.0
var p =
[
  {
    "id":100,
    "name":"Saikiran",
    "Org":"Cognizant"
  }
]
output application/json
---
```

```
[{"The value is": 100}, {"The value is": "Saikiran"}, {"The value is": "Cognizant"}]
```

# Pluck with joinBy

This screenshot shows a DataWeave transformation in the DataWeave Playground. The payload is a JSON object with an array of skills. The script uses the `pluck` function to extract the skill names into an array. A red circle highlights the cursor position at the end of the `pluck` function call.

```
1 < PAYLOAD          JSON
2   {
3     "id": 101,
4     "name": "Saikiran",
5     "location": "Vizag",
6     "Skill": [
7       {
8         "primaryskill": "Mulesoft",
9         "secondaryskill": "Java",
10        "Otherskill": "JavaScript"
11      }
12   }
13
14 < SCRIPT
15   %dw 2.0
16
17   output application/json
18
19   ---
20
21   {
22     "employeeID": payload.id,
23     "employeeName": payload.name,
24     "employeeLoc": payload.location,
25     "employeeSkills": payload.Skill pluck $ |
```

```
1 < OUTPUT          JSON
2   {
3     "employeeID": 101,
4     "employeeName": "Saikiran",
5     "employeeLoc": "Vizag",
6     "employeeSkills": [
7       "Mulesoft",
8       "Java",
9       "JavaScript"
10    ]
11 }
```

DataWeave Playground

Export Import Tutorial Playground

This screenshot shows a DataWeave transformation in the DataWeave Playground. The payload is the same as the first example. The script uses the `pluck` function with a `joinBy` argument to join the skill names into a single string. A red box highlights the `joinBy` argument. Another red box highlights the resulting concatenated string in the output.

```
1 < PAYLOAD          JSON
2   {
3     "id": 101,
4     "name": "Saikiran",
5     "location": "Vizag",
6     "Skill": [
7       {
8         "primaryskill": "Mulesoft",
9         "secondaryskill": "Java",
10        "Otherskill": "JavaScript"
11      }
12   }
13
14 < SCRIPT
15   %dw 2.0
16
17   output application/json
18
19   ---
20
21   {
22     "employeeID": payload.id,
23     "employeeName": payload.name,
24     "employeeLoc": payload.location,
25     "employeeSkills": payload.Skill pluck $ joinBy "," |
```

```
1 < OUTPUT          JSON
2   {
3     "employeeID": 101,
4     "employeeName": "Saikiran",
5     "employeeLoc": "Vizag",
6     "employeeSkills": "Mulesoft,Java,JavaScript"
7   }
```

# Operator: Flatten

Flatten combine all the data and gives the required data in single set of array

The image shows two side-by-side screenshots of the Mule Studio IDE interface. Both windows have a title bar with tabs for 'Output Payload' and 'Preview'. The left window is titled 'ems' and contains the following code:

```
16   "name":"Sai",
17   "Org":"Cognizant"
18 }
19 ]
20
21=var h =[

22{
23  "id":10011,
24  "name":"Saikiran",
25  "Org":"Cognizant"
26},
27{
28  "id":1013,
29  "name":"kiran",
30  "Org":"Cognizant"
31},
32{
33  "id":1023,
34  "name":"Sai",
35  "Org":"Cognizant"
36}
37]
38
39 output application/json
40 ---
41 flatten(p+h).id
```

The right window is titled 'problems' and contains the following code:

```
1=%dw 2.0
2
3=var p =[

4{
5  "id":100,
6  "name":"Saikiran",
7  "Org":"Cognizant"
8},
9{
10  "id":101,
11  "name":"kiran",
12  "Org":"Cognizant"
13},
14{
15  "id":103,
16  "name":"Sai",
17  "Org":"Cognizant"
18}
19]
20
21 output application/json
22 ---
23 flatten(p|.id)
```

Both windows show a preview pane at the bottom with tabs for 'Tree' and 'Text'. The 'Text' tab is selected in both.

Flatten: Used to merge 2 or more array of objects/Objects into single array

- Difference by using flatten keyword & Without flatten keyword

	OUTPUT	JSO
%dw 2.0	[ ]	[ ]
var arr1= [ {	{	{
"id": 101,	"id": 101,	"id": 101,
"name": "Saikiran",	"name": "Saikiran",	"name": "Saikiran",
"location": "Vizag"	"location": "Vizag"	"location": "Vizag"
},	},	},
{	{	{
"id": 102,	"id": 102,	"id": 102,
"name": "Pragna",	"name": "Pragna",	"name": "Pragna",
"location": null	"location": null	"location": null
}]	},	},
var arr2 =	[	[
[{	{	{
"id": 103,	"id": 103,	"id": 103,
"name": "Hero",	"name": "Hero",	"name": "Hero",
"location": "Pvp"	"location": "Pvp"	"location": "Pvp"
}	},	},
	]	]
var arr3 = arr1+arr2		
output application/json		
---		
flatten(arr3)]		

	SCRIPT	OUTPUT
1 %dw 2.0	1 %dw 2.0	1 [ ]
2 var arr1= [ {	2 var arr1= [ {	2 {
3   "id": 101,	3   "id": 101,	3   "id": 101,
4   "name": "Saikiran",	4   "name": "Saikiran",	4   "name": "Saikiran",
5   "location": "Vizag"	5   "location": "Vizag"	5   "location": "Vizag"
6 },	6 },	6 },
7 {	7 {	7 {
8   "id": 102,	8   "id": 102,	8   "id": 102,
9   "name": "Pragna",	9   "name": "Pragna",	9   "name": "Pragna",
10   "location":null	10   "location":null	10   "location":null
11 }]	11 }]	11 },
12 var arr2 =	12 var arr2 =	12 [
13 [{	13 [{	13 {
14   "id": 103,	14   "id": 103,	14   {id": 103,
15   "name": "Hero",	15   "name": "Hero",	15   "name": "Hero",
16   "location": "Pvp"	16   "location": "Pvp"	16   "location": "Pvp"
17 }]	17 }]	17 }
18	18	18 ]
19 var arr3 = arr1+arr2	19 var arr3 = arr1+arr2	19 [ ]
20 output application/json	20 output application/json	20 [ ]
---	---	21 [ ]
22 arr3	22 arr3	22 [ ]
23	23	23 [ ]
24	24	24 [ ]

without flatten

# Filter Key in DataWeave

If we want to filter by key, value

-> eg : \$.id>200

**Note:** We use filter for map( ) and filterObject for mapObject( )

The screenshot shows a DataWeave editor interface. On the left, the 'Output Payload' pane displays the following JSON code:

```
12 {  
13   "id":103,  
14   "name":"Datti",  
15   "Org":"Cognizant"  
16 },  
17 {  
18   "id":10011,  
19   "name":"Venki",  
20   "Org":"Cognizant"  
21 },  
22 {  
23   "id":1013,  
24   "name":"Kiran",  
25   "Org":"Cognizant"  
26 },  
27 {  
28   "id":1023,  
29   "name":"Ashok",  
30   "Org":"Cognizant"  
31 }  
32  
33 ]  
34  
35 output application/json  
36 ---  
37 p map{  
38   id:$ .id,  
39   name:$ .name,  
40   Org:$ .Org  
41 }filter $ .id > 200|
```

On the right, the 'Output' pane shows the resulting JSON array:

```
[  
  {  
    "id": 10011,  
    "name": "Venki",  
    "Org": "Cognizant"  
  },  
  {  
    "id": 1013,  
    "name": "Kiran",  
    "Org": "Cognizant"  
  },  
  {  
    "id": 1023,  
    "name": "Ashok",  
    "Org": "Cognizant"  
  }  
]
```

# groupBy in DataWeave

groupBy is used to group the array according to the key(id, name,...)

```
12@ {  
13@   "id":103,  
14@   "name":"Datti",  
15@   "Org":"Cognizant"  
16@ },  
17@ {  
18@   "id":10011,  
19@   "name":"Venki",  
20@   "Org":"Cognizant"  
21@ },  
22@ {}  
23@ {  
24@   "id":1013,  
25@   "name":"kiran",  
26@   "Org":"Cognizant"  
27@ },  
28@ {  
29@   "id":1023,  
30@   "name":"Ashok",  
31@   "Org":"Cognizant"  
32@ }  
33 ]  
34  
35 output application/json  
36 ---  
37@p map{  
38@   id:$.id,  
39@   name:$.name,  
40@   Org:$Org  
41 }groupBy $.id
```

By ID

```
Output Payload ▾ ⌂ ↻ 🔍 111 111  
12@ {  
13@   "id":103,  
14@   "name":"Datti",  
15@   "Org":"Cognizant"  
16@ },  
17@ {  
18@   "id":10011,  
19@   "name":"Venki",  
20@   "Org":"Cognizant"  
21@ },  
22@ {  
23@   "id":1013,  
24@   "name":"kiran",  
25@   "Org":"Cognizant"  
26@ },  
27@ {  
28@   "id":1023,  
29@   "name":"Ashok",  
30@   "Org":"Cognizant"  
31@ }  
32  
33 ]  
34  
35 output application/json  
36 ---  
37@p map{  
38@   id:$id,  
39@   name:$name,  
40@   Org:$Org  
41 }
```

By name

# orderBy in DataWeave

orderBy is used to keep the array in Order (Ascending/Descending)

Output Payload ▾ ⌂ ↻ 1 issue found

```
12{  
13  "id":103,  
14  "name":"Datti",  
15  "Org":"Cognizant"  
16 },  
17 {  
18  "id":10011,  
19  "name":"Venki",  
20  "Org":"Cognizant"  
21 },  
22 {  
23  "id":1013,  
24  "name":"kiran",  
25  "Org":"Cognizant"  
26 },  
27 {  
28  "id":1023,  
29  "name":"Ashok",  
30  "Org":"Cognizant"  
31 }  
32  
33 ]  
34  
35 output application/json  
36 ---  
37@p map{  
38  id:$ .id,  
39  name:$ .name,  
40  Org:$ .Org  
41 }orderBy $.id|
```

**Id**

Output Payload ▾ ⌂ ↻ 1 issue found

```
12{  
13  "id":103,  
14  "name":"Datti",  
15  "Org":"Cognizant"  
16 },  
17 {  
18  "id": 101,  
19  "name": "Saikiran",  
20  "Org": "Cognizant"  
21 },  
22 {  
23  "id": 103,  
24  "name": "Datti",  
25  "Org": "Cognizant"  
26 },  
27 {  
28  "id": 1013,  
29  "name": "kiran",  
30  "Org": "Cognizant"  
31 },  
32 {  
33  "id": 1023,  
34  "name": "Ashok",  
35  "Org": "Cognizant"  
36 }  
37@p map{  
38  id:$ .id,  
39  name:$ .name,  
40  Org:$ .Org  
41 }
```

**Name**

## distinctBy : To remove duplicate records (Unique date)

PAYLOAD	JSON	SCRIPT	OUTPUT	JSON
<pre>1  [{ 2    "id": 101, 3    "name":"Saikiran", 4    "location":"Vizag" 5  }, 6  { 7    "id": 102, 8    "name":"Pragna" 9  }, 10 { 11   "id": 103, 12   "name":"Villan" 13 }, 14 { 15   "id": 103, 16   "name":"Hero", 17   "location":"Pvp" 18 }, 19 { 20   "id": 104, 21   "name":"Ramu", 22   "location":"BBL" 23 }]</pre>		<pre>1  %dw 2.0 2  output application/json 3  --- 4  payload map{ 5    "Employee ID": \$.id, 6    "Location":\$.location 7  } distinctBy \$.Employee ID 8 9 10</pre>	<pre>1  [ 2    { 3      "Employee ID": 101, 4      "Location": "Vizag" 5    }, 6    { 7      "Employee ID": 102, 8      "Location": null 9    }, 10   { 11     "Employee ID": 103, 12     "Location": null 13   }, 14   { 15     "Employee ID": 104, 16     "Location": "BBL" 17   } 18 ]</pre>	

The screenshot shows a Mule ETL interface with four panels: PAYLOAD, JSON, SCRIPT, and OUTPUT. The PAYLOAD panel contains a list of employee records. The JSON panel shows the same list in JSON format. The SCRIPT panel contains a Mule configuration script using the %dw 2.0 language. It defines a payload map where each record is converted to a map with 'Employee ID' and 'Location' keys. The 'distinctBy' function is used on the map to remove duplicates based on the 'Employee ID'. The OUTPUT panel shows the resulting list, which contains four unique records. Red circles and arrows highlight the 'id' field in the PAYLOAD and the 'Employee ID' field in the SCRIPT and OUTPUT, illustrating how the unique key is mapped from the source to the output.

# default : Not to store Null values

If the id/location/name is empty  
then a default placeholder will be  
set in that place

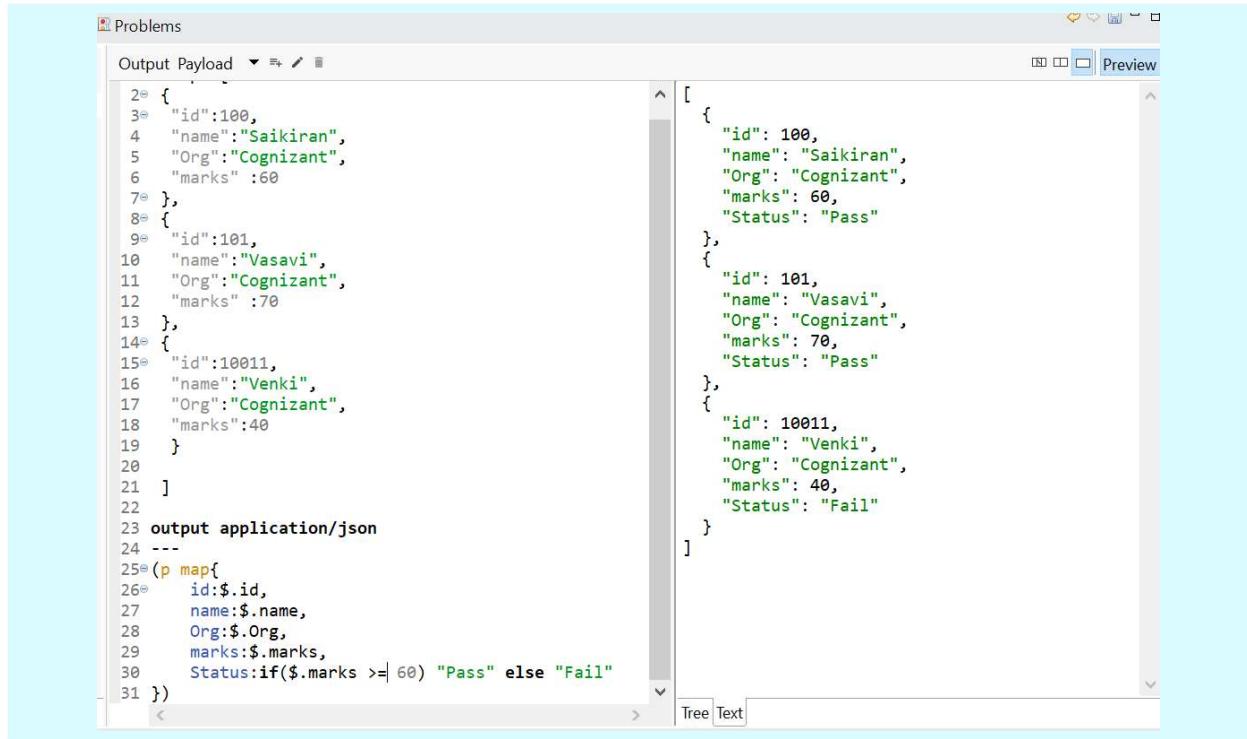
EXPERIENCE INNOVATION, UNLEASHED. WATCH THE HIGHLIGHTS FROM CONNECT '22

DataWeave Playground

PAYOUT	JSON	SCRIPT	OUTPUT	JSON
<pre>1 [{ 2   "id": 101, 3   "name":"Saikiran", 4   "location":"Vizag" 5 }, 6 { 7   "id": 102, 8   "name":"Pragna", 9   "location":null 10 }, 11 { 12   "id": 103, 13   "name":"Hero", 14   "location":"Pvp" 15 }, 16 { 17   "id": 104, 18   "name":"Ramu", 19   "location":null 20 }]</pre>		<pre>1 %dw 2.0 2 output application/json 3 --- 4 payload map{ 5   "Employee ID": \$.id, 6   EmployeeName:\$ .name, 7   "Location":\$.location default "----" 8 } distinctBy \$.Employee ID 9 10 11</pre>	<pre>1 [ 2   { 3     "Employee ID": 101, 4     "EmployeeName": "Saikiran", 5     "Location": "Vizag" 6   }, 7   { 8     "Employee ID": 102, 9     "EmployeeName": "Pragna", 10    "Location": "----" 11   }, 12   { 13     "Employee ID": 103, 14     "EmployeeName": "Hero", 15     "Location": "Pvp" 16   }, 17   { 18     "Employee ID": 104, 19     "EmployeeName": "Ramu", 20     "Location": "----" 21   } 22 ]</pre>	

The screenshot shows a DataWeave playground interface. The payload contains four employee records. The fourth record has a null value for the location field. In the script editor, a red arrow points to the line where the location field is mapped to a default value of "----". The resulting output JSON shows that the location field for the fourth employee is now "----" instead of null.

# If-else in DataWeave



The screenshot shows a DataWeave editor interface. On the left, the "Output Payload" pane displays the following JSON code:

```
20 {  
21   "id":100,  
22   "name": "Saikiran",  
23   "Org": "Cognizant",  
24   "marks" :60  
25 },  
26 {  
27   "id":101,  
28   "name": "Vasavi",  
29   "Org": "Cognizant",  
30   "marks" :70  
31 },  
32 {  
33   "id":10011,  
34   "name": "Venki",  
35   "Org": "Cognizant",  
36   "marks":40  
37 }  
38 ]  
39  
40 output application/json  
41 ---  
42 (p map{  
43   id:$ .id,  
44   name:$ .name,  
45   Org:$ .Org,  
46   marks:$ .marks,  
47   Status:if($.marks >= 60) "Pass" else "Fail"  
48 })
```

On the right, the "Preview" pane shows the resulting JSON output:

```
[  
  {  
    "id": 100,  
    "name": "Saikiran",  
    "Org": "Cognizant",  
    "marks": 60,  
    "Status": "Pass"  
  },  
  {  
    "id": 101,  
    "name": "Vasavi",  
    "Org": "Cognizant",  
    "marks": 70,  
    "Status": "Pass"  
  },  
  {  
    "id": 10011,  
    "name": "Venki",  
    "Org": "Cognizant",  
    "marks": 40,  
    "Status": "Fail"  
  }]
```

- If-else is a conditional statement
- Syntax:  
Eg: Status: if(\$.marks >= 60)  
"Pass" else "Fail"

# Task:- Learn about DataWeave Functions and come back

- Link to Learn Functions/Methods:

- [https://www.w3schools.com/java/java\\_methods.asp](https://www.w3schools.com/java/java_methods.asp)

The screenshot shows the Mule Studio IDE interface. The left panel displays a DataWeave script with line numbers 1 through 45. The right panel shows the resulting JSON output. The DataWeave script defines a variable 'a' as a list of three candidate objects: Sravan, Mule, and Swag. Each candidate has an ID, name, and subject-wise details (Maths, Physics, English) with their respective marks and status (PASS or FAIL). The script uses functions like 'if' and 'else' to determine the status based on marks.

```
1 //dwv 2.8
2 output application/json
3@ var a = [
4@   {
5@     name : "Sravan",
6@     id :123,
7@     "Maths" : 25,
8@     "Physics" : 56,
9@     "English" : 72
10@   },
11@   {
12@     name : "Mule",
13@     id :456,
14@     "Maths" : 85,
15@     "Physics" : 46,
16@     "English" : 62
17@   },
18@   {
19@     name : "Swag",
20@     id :786,
21@     "Maths" : 85,
22@     "Physics" : 76,
23@     "English" : 52
24@   }
25 ]
26
27
28@ a map {
29
30@   "Candidate" : $.name,
31@   "RollNo" : $.id,
32@   "SubjectWiseDetails" : {
33@     "Maths": {marks : $.Maths , status : if($.Maths > 50) "PASS" else "FAIL"} ,
34@     "Physics": {marks : $.Physics , status : if($.Physics > 50) "PASS" else "FAIL"} ,
35@     "English": {marks : $.English , status : if($.English > 50) "PASS" else "FAIL"} ,
36@   }
37
38 }
39
40
41 //show usage of function , function with arguments with data types
42
43
```

```
[{"Candidate": "Sravan", "RollNo": 123, "SubjectWiseDetails": {"Maths": {"marks": 25, "status": "FAIL"}, "Physics": {"marks": 56, "status": "PASS"}, "English": {"marks": 72, "status": "PASS"}}, {"Candidate": "Mule", "RollNo": 456, "SubjectWiseDetails": {"Maths": {"marks": 85, "status": "PASS"}, "Physics": {"marks": 46, "status": "FAIL"}, "English": {"marks": 62, "status": "FAIL"}}, {"Candidate": "Swag", "RollNo": 786, "SubjectWiseDetails": {"Maths": {"marks": 85, "status": "PASS"}, "Physics": {"marks": 76, "status": "FAIL"}, "English": {"marks": 52, "status": "FAIL"}}}
```

The screenshot shows the Mule Studio IDE interface. The left panel displays a DataWeave script with line numbers 1 through 34. The right panel shows the resulting JSON output. The DataWeave script defines a variable 'a' as a list of three candidate objects: Sravan, Mule, and Swag. Each candidate has an ID, name, and subject-wise details (Maths, Physics, English) with their respective marks. The script includes a function 'checkResult(marks)' which returns 'PASS' if marks are greater than 50, and 'FAIL' otherwise. The output shows the same data as the previous screenshot, but with the 'checkResult' function applied to the 'Maths' marks.

```
1 //dwv 2.8
2 output application/json
3@ var a = [
4@   {
5@     name : "Sravan",
6@     id :123,
7@     "Maths" : 25,
8@     "Physics" : 56,
9@     "English" : 72
10@   },
11@   {
12@     name : "Mule",
13@     id :456,
14@     "Maths" : 85,
15@     "Physics" : 46,
16@     "English" : 62
17@   },
18@   {
19@     name : "Swag",
20@     id :786,
21@     "Maths" : 85,
22@     "Physics" : 76,
23@     "English" : 52
24@   }
25 ]
26
27 fun checkResult(marks) = if(marks > 50) "PASS" else "FAIL"
28
29
30 checkResult("53")
31
32 //show usage of function , function with arguments with data types
33
34
```

```
[{"Candidate": "Sravan", "RollNo": 123, "SubjectWiseDetails": {"Maths": {"marks": 25, "status": "FAIL"}, "Physics": {"marks": 56, "status": "PASS"}, "English": {"marks": 72, "status": "PASS"}}, {"Candidate": "Mule", "RollNo": 456, "SubjectWiseDetails": {"Maths": {"marks": 85, "status": "PASS"}, "Physics": {"marks": 46, "status": "FAIL"}, "English": {"marks": 62, "status": "FAIL"}}, {"Candidate": "Swag", "RollNo": 786, "SubjectWiseDetails": {"Maths": {"marks": 85, "status": "PASS"}, "Physics": {"marks": 76, "status": "FAIL"}, "English": {"marks": 52, "status": "FAIL"}}}]
```

# TypeCasting

The image shows two side-by-side Fiddler-style tool windows for inspecting API responses.

**Left Window (Output Payload):**

```
3@ var v1={  
4@   "id": "101", ✓  
5@   "name": "Saikiran",  
6@   "location": "Vizag",  
7@   "primaryskill": "Mulesoft",  
8@   "isActive": "true" ✓  
9@ }  
11 ---  
12@ {  
13@   "studentID": v1.id as Number, ✓  
14@   "Status": v1.isActive as Boolean, ✓  
15@   "studentName": v1.name  
16@ }
```

**Right Window (Output Payload):**

```
3@ var v1={  
4@   "id": "101", ✓  
5@   "name": "Saikiran",  
6@   "location": "Vizag",  
7@   "primaryskill": "Mulesoft",  
8@   "isActive": "true" ✓  
9@ }  
10@ }  
11 ---  
12@ {  
13@   "studentID": v1.id ✓  
14@   "Status": v1.isActive, ✓  
15@   "studentName": v1.name  
16@ }
```

**Left Window (Preview):**

Loading Data ...

- Object
  - studentID : Number = 101 ✓
  - Status : Boolean = true ✓
  - studentName : String = "Saikiran"

**Right Window (Preview):**

Loading Data ...

- Object
  - studentID : String = "101"
  - Status : String = "true"
  - studentName : String = "Saikiran"

## Get Time with now(),now() as Date & Required format

Now()	“YYYY-MM-DDTHH:SS:MS.SSSZ”
now() as Date	“YYYY:MM:DD”
Now() as Date as String{format:'dd-MM-YY'}	“DD-MM-YY”

SCRIPT

```
1 %dw 2.0
2
3 output application/json
4 ---
5 "Date": now() as String{format:'dd-MM-YY'}
```

OUTPUT

```
1 [
2   "Date": "27-09-22"
3 ]
```

SCRIPT

```
1 %dw 2.0
2
3 output application/json
4 ---
5 now()
```

OUTPUT

```
1 "2022-09-27T14:12:52.694263Z"
```

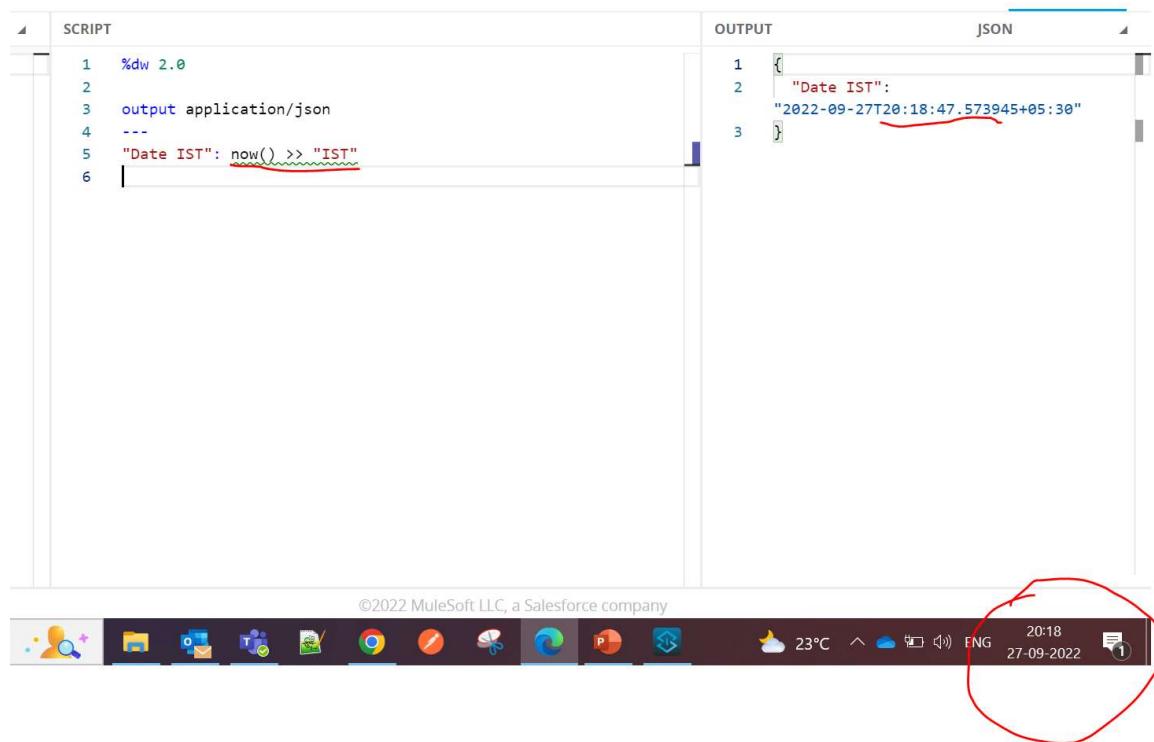
SCRIPT

```
1 %dw 2.0
2
3 output application/json
4 ---
5 now() as Date
```

OUTPUT

```
1 "2022-09-27"
YY:MM:DD
```

# Date Country Wide



The screenshot shows a desktop interface with a script editor and a system tray.

**Script Editor:**

```
SCRIPT
1 %dw 2.0
2
3 output application/json
4 ---
5 "Date IST": now() >> "IST"
6
```

**Output:**

```
OUTPUT JSON
1 {
2   "Date IST":
3     "2022-09-27T20:18:47.573945+05:30"
4 }
```

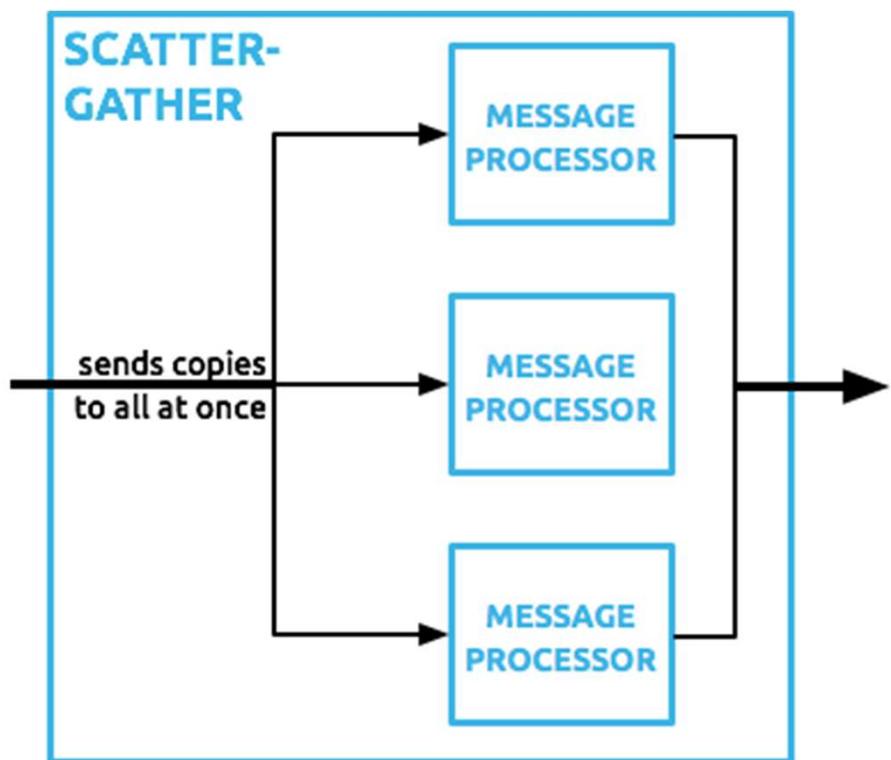
A red circle highlights the system tray icon in the bottom right corner of the desktop bar.

# DataWeave Lookup

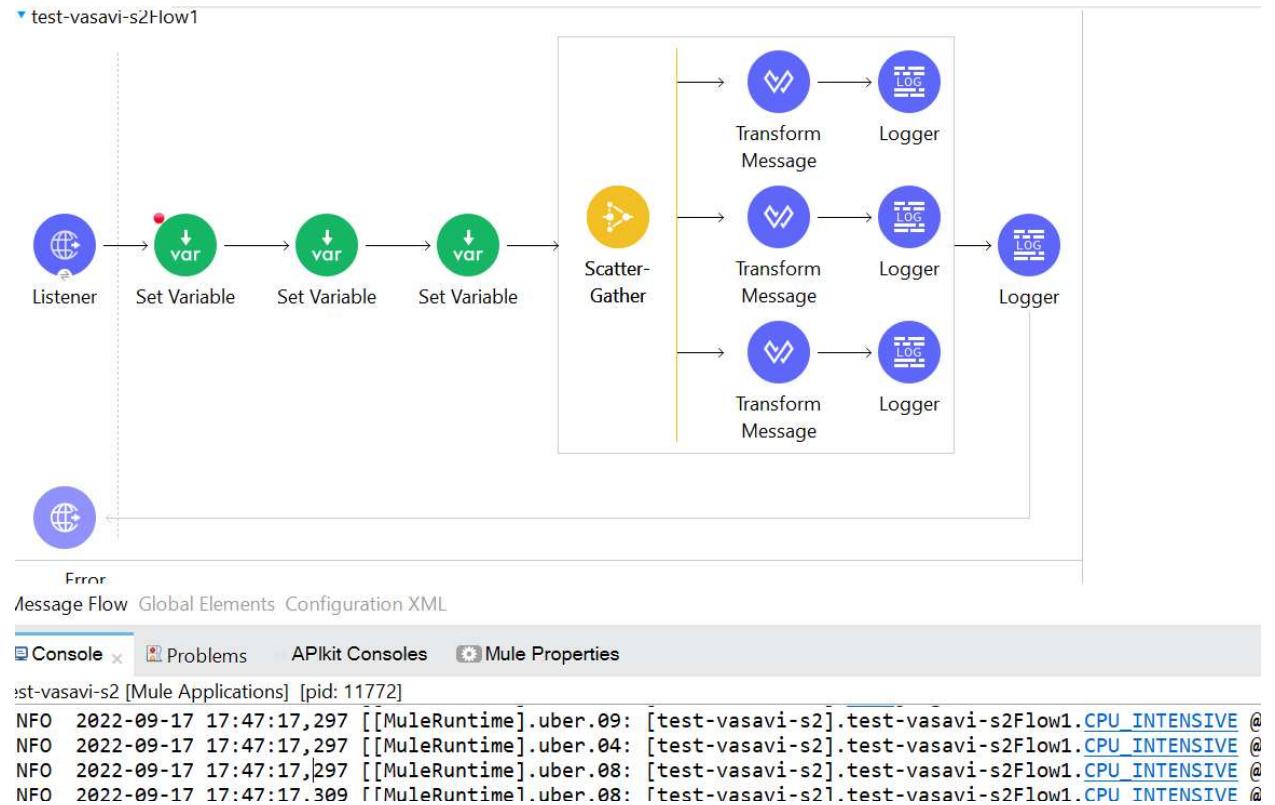
- To call any “**private flow**” or Subflow we are using Flow reference
- We can also call private flow component with DataWeave function called Lookup
- We cannot call Subflow with with DataWeave function “Lookup”
  - Syntax: Mule::lookup(“private-flow-name”, “Payload can be passed here”)
  - Link:  
<https://docs.mulesoft.com/dataweave/2.2/dw-mule-functions-lookup>

# Scatter-Gather

- Scatter Gather sends the request message to multiple targets concurrently, It collects responses from various targets and aggregate them to a single message
- Output payload is the combination all targets. And each target output has (Payload+attribute)
- So its best practice to wrap every target within Try-Block with On-Error-Continue, So you can handle the errors and complete the process successfully
- The total time taken to process all targets is the time took by single process (i.e Max 20secs)
- Use Try, On error continue to avoid errors. Because if you get error even in one process, all processes will be stopped

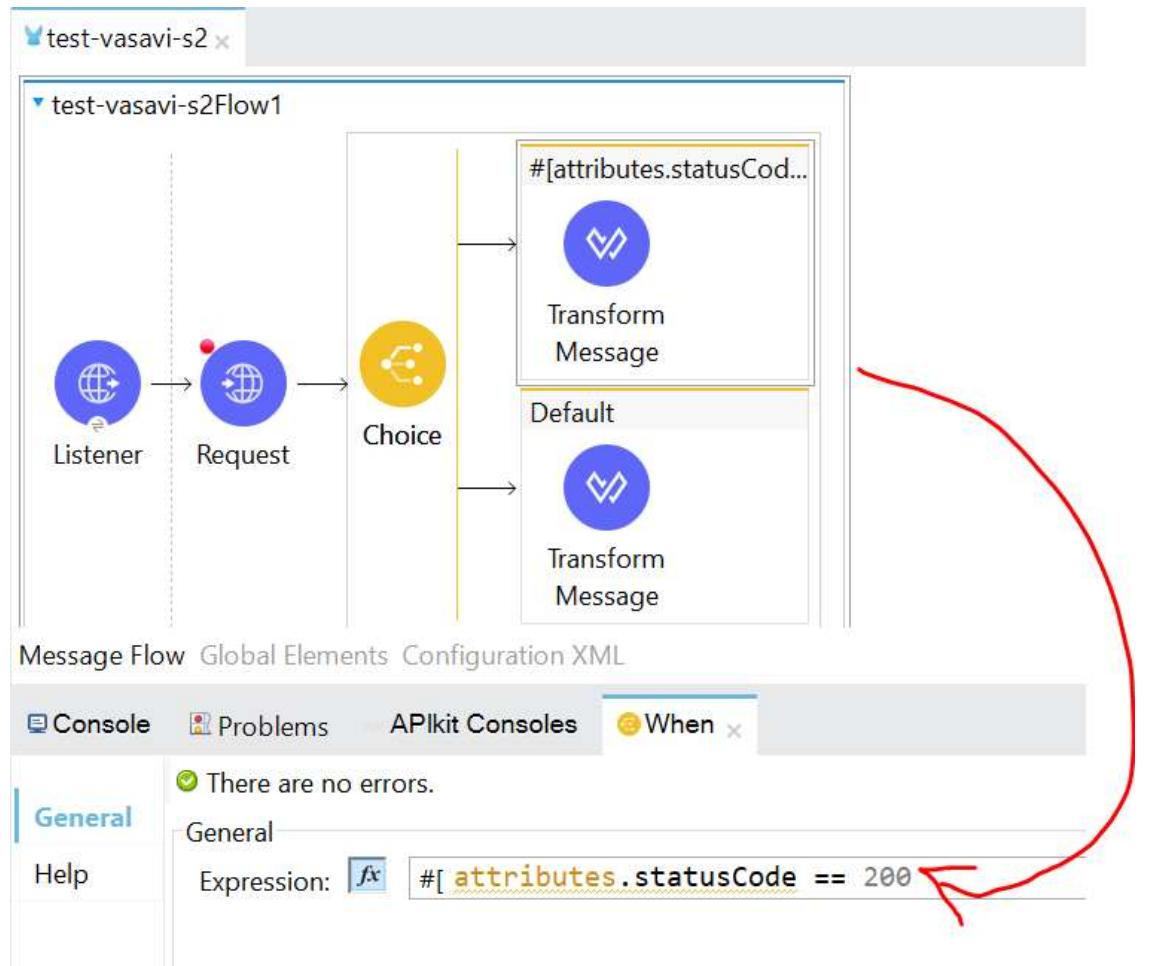


# Scatter Gather: Example

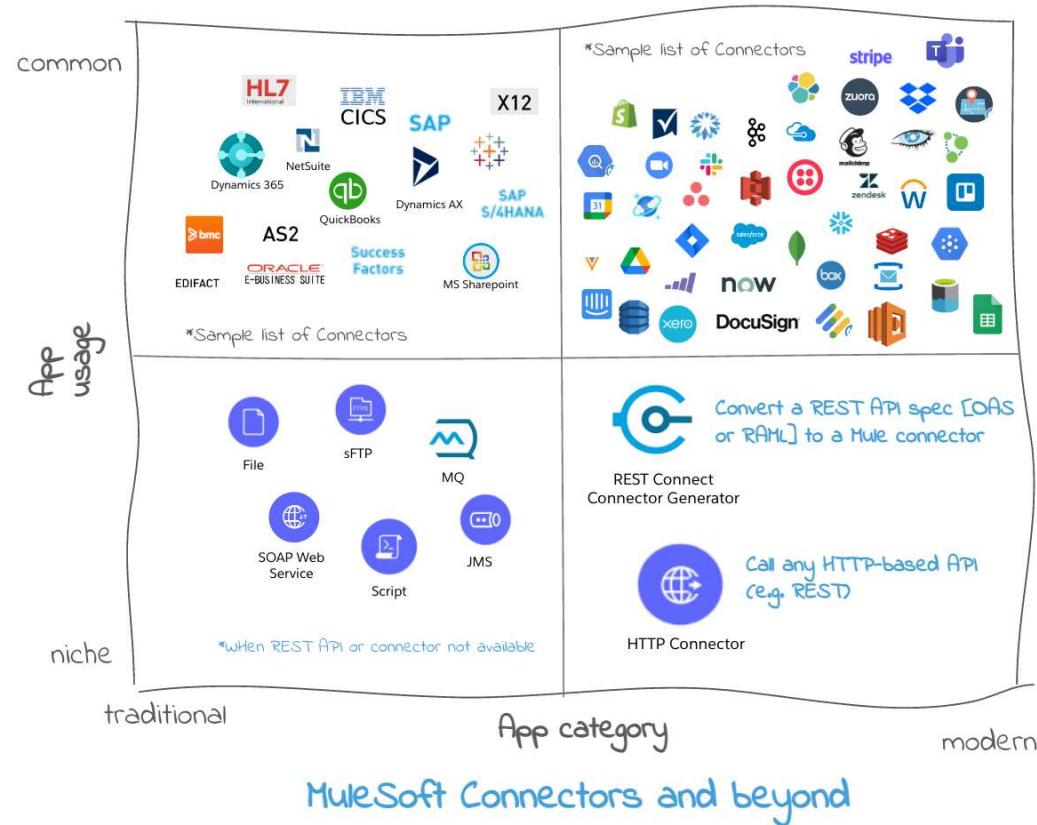


# Choice Reutter

- The Choice router dynamically routes messages through a flow according to a set of DataWeave expressions that evaluate message content
- Choice-reutter is like if-else condition. It checks the positive and negative conditions and respond accordingly



# Connectors



# Important Thing to remember while working with connectors

File,SFTP, Database, Salesforce, any External web service etc... are some connectors

Gather the **configuration details** that are required while working with specific connector

See what is the **request** that particular system is **accepting** (Query param, Uri path, etc..)

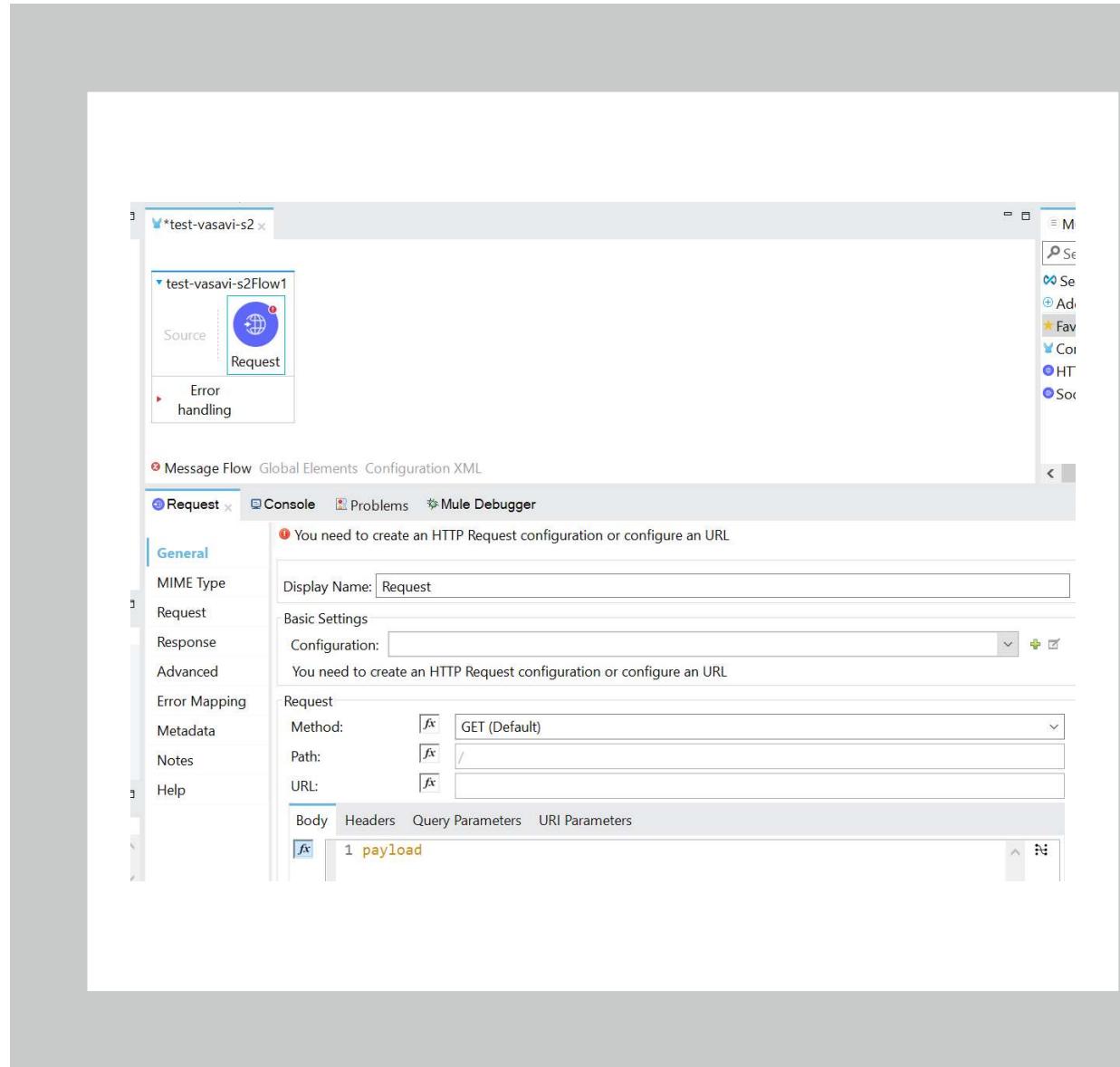
Check security that required to connect external system

Check what type of response returns back

Check what amount of time taking to get the response for particular request

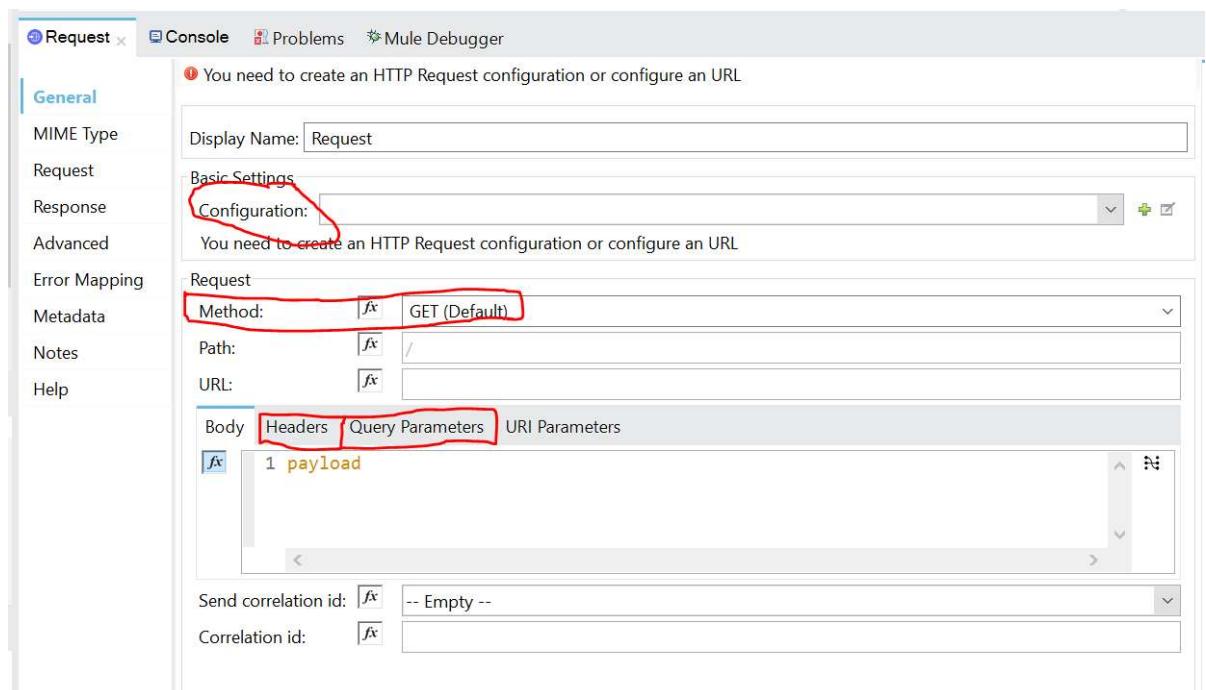
# Connector: Request

- In http request, We are giving control to another system (External) or API and requesting them to give control over that system
- Http Request is used to call other Web-services
- Response is received from External system
- It is placed in “Process” area of private flow (or) Subflow
- We need to make sure on sending proper inputs to http Request components like URL, method, the input payload and attributes like query and uri param what ever web-service is expecting



# Request : Mandatory Config details

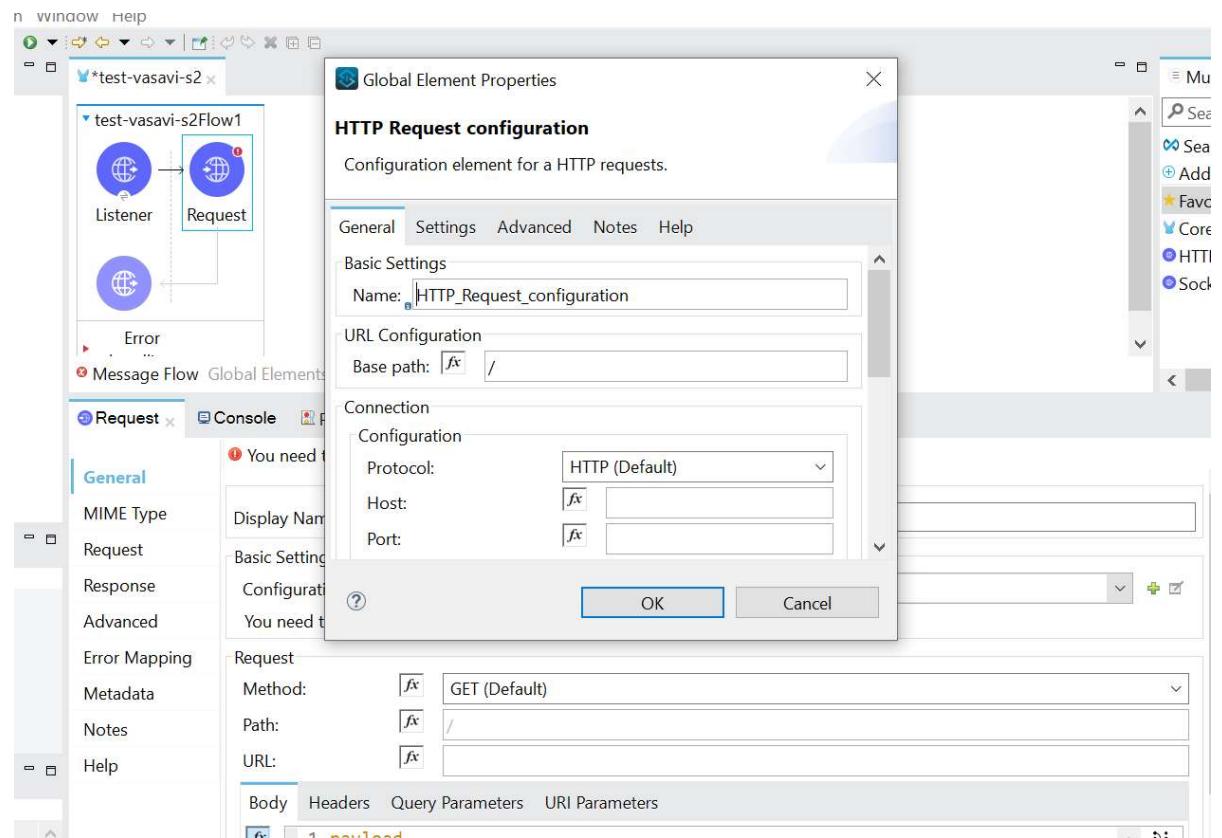
- Mandatory config details
  - Protocol (Http/https)
  - Method (Get/Post/Delete/Put)
  - URL or Path : If you provide whole URL, Then port base path are not required. If you give path, then host, port, path are necessary
- You can pass Body, headers, attributes, queryParams in respective tabs
- In the case of **Error** coming from Web-service, we are calling to be considered as success, then have make changes in response tab of **Http Request** and make use of **Success-status-code-validator**



# Some External API's we can use

---

- Weather API:
- <https://openweathermap.org/api#current>
- Employees Data:
- <https://dummy.restapiexample.com/api/v1/employees>
- Configuring External API



# Response of SetRequest is same as Rest API

The image displays two side-by-side screenshots of the Postman application interface, illustrating that the response from a SetRequest is identical to that of a standard REST API.

**Left Screenshot (SetRequest Response):**

- Method: GET
- URL: <http://localhost:8081/test>
- Body tab selected
- Response status: 200 OK
- Response time: 7.26 s
- Response size: 28.87 KB
- Pretty JSON response:

```
1 "cod": "200",
2 "message": 0,
3 "cnt": 40,
4 "list": [
5   {
6     "dt": 1663416000,
7     "main": {
8       "temp": 289.46,
9       "feels_like": 289,
10      "temp_min": 289.46,
11      "temp_max": 289.71,
12      "pressure": 1002,
```

**Right Screenshot (Rest API Response):**

- Method: GET
- URL: <http://api.openweathermap.org/data/2.5/forecast?id=524901&appid=5570a62af30e8a9b35526331fe...>
- Body tab selected
- Response status: 200 OK
- Response time: 224 ms
- Response size: 16.21 KB
- Pretty JSON response:

```
1
2 "cod": "200",
3 "message": 0,
4 "cnt": 40,
5 "list": [
6   {
```

# Connector : Validation

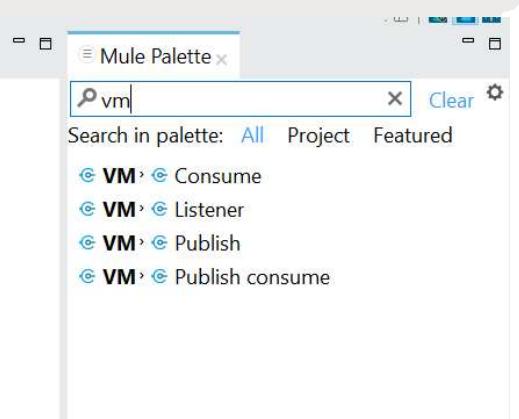
Validation component verifies the content of the message whether it matches the specific criteria

-> There are a lot of validation connectors as listed below

The screenshot shows the Mule Studio interface with three main panels:

- Mule Palette:** A sidebar on the left containing a search bar for "validation". Below it is a list of validation components:
  - Validation Is IP
  - Validation Is URL
  - Validation Is allowed ip
  - Validation Is blank string
  - Validation Is elapsed
  - Validation Is email
  - Validation Is empty collection
  - Validation Is false
  - Validation Is not blank string
  - Validation Is not denied ip
  - Validation Is not elapsed
  - Validation Is not empty collection
  - Validation Is not null
  - Validation Is null
- Message Flow:** The central workspace displays a flow named "test-vasavi-s2Flow1". The flow consists of the following components connected sequentially: Listener → Set Payload → Is not null → Transform Message → Logger. An "Error handling" section is also present.
- Set Payload Component Configuration:** A detailed view of the "Set Payload" component. It shows the "General" tab with the following settings:
  - Display Name: Set Payload
  - Value: #[ null]Below this, the "Params" tab is shown for a GET request to "http://localhost:8081/test", with a single parameter "KEY" set to "VALUE".

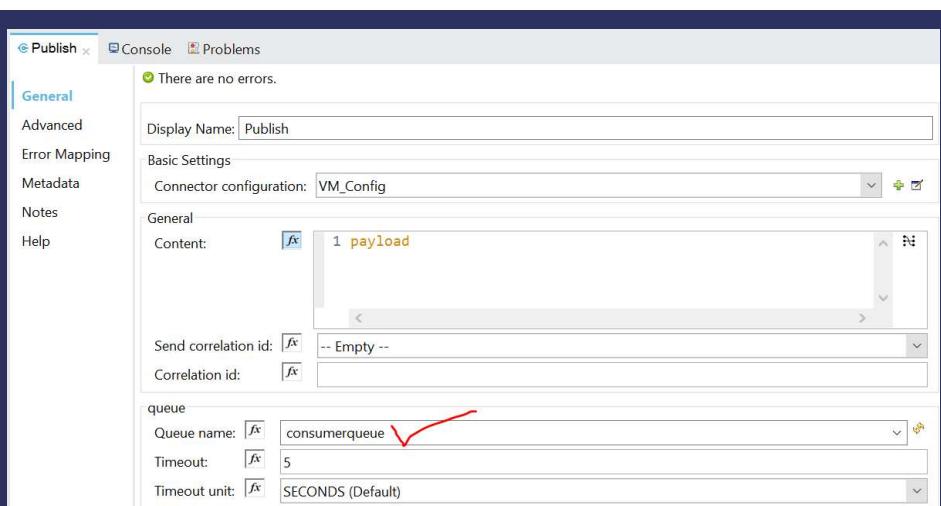
# VM Connector



- Virtual Machine (VM Connector) manages intra-app and inter-app communication through either transient or persistent asynchronous queues
  - **Transient Queues:** Faster than persistent queues, But they are not reliable if system crashes
  - **Persistent Queues :** Slower but reliable
- **What is Transient Queue?**
  - Transient Queue is used to make process Faster to store data in the memory, But they are not reliable if system crashes
  - Transient Queue is faster than persistent Queue
- **What is Persistent Queue?**
  - Persistent Queues are slower while compared with the transient queues to store data in the memory, But they are reliable
  - Persistent Queue is Slower than Transient Queue
- **When we will use VM connector**
  - If we want to pass the messages to another flow through queue mechanism instead of Flow-ref, We will go with VM Connector
  - Used Communicate between different apps with in Mule system
  - When we want to distribute work across the system

# VM connector: Publish, Consume, Listener, Publish consume

- Publish : Used to publish the message to queue and receive message from queue
- Consume: Used to retrieve/Subscribe the data from Publish after the end of the main flow
- Publish Consume: Used in place of Publish to retrieve/Subscribe the data from subscriber flow
- Note: Using **Post** method



# VM Config

**Global Element Properties**

**VM Config**  
Default configuration

General Advanced Notes Help

Connection  
 Reconnection  
 Fails deployment when test connection fails  
Reconnection strategy: None

General  
Queues  
Queues **Edit inline**

Queue name	Queue type	Max outstanding ...
consumerqueue	TRANSIENT	0

OK Cancel

**Global Element Properties**

**Queue**

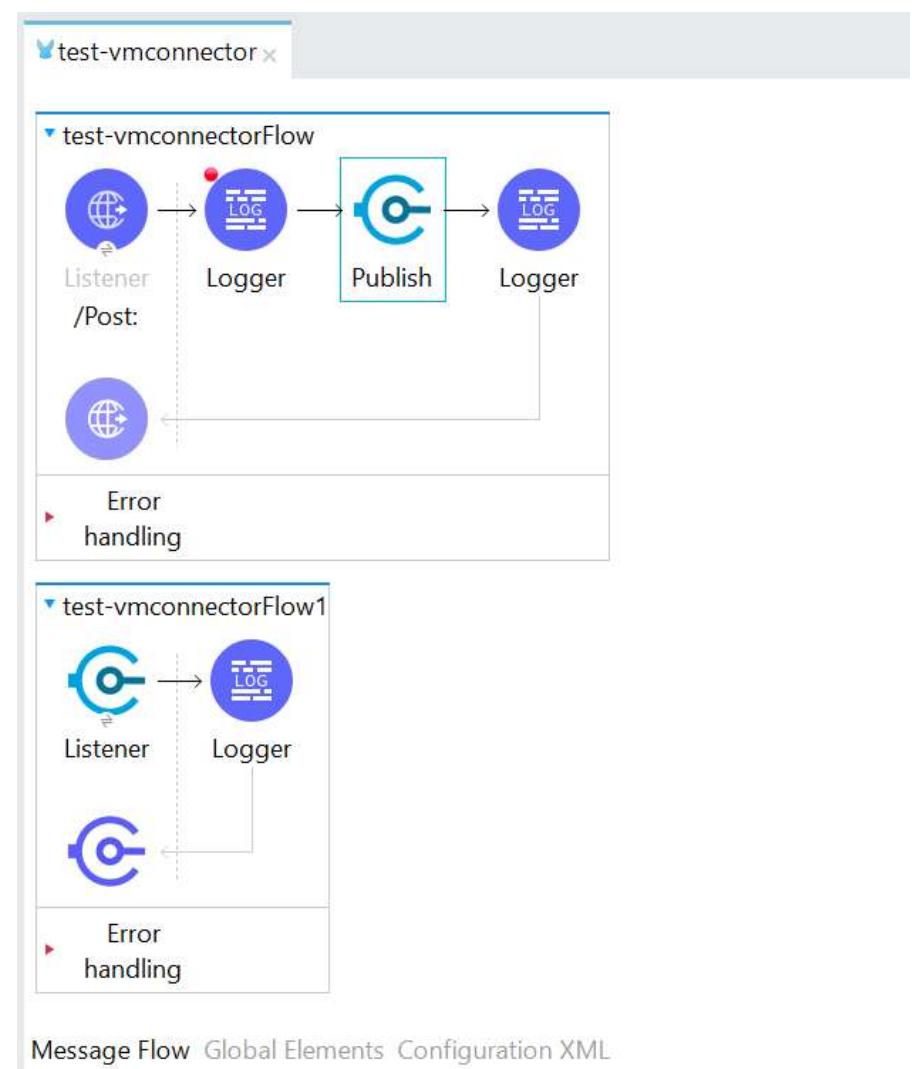
Queue  
Queue name: consumerqueue  
Queue type: TRANSIENT (Default)

Finish Cancel

**Global Element Properties**

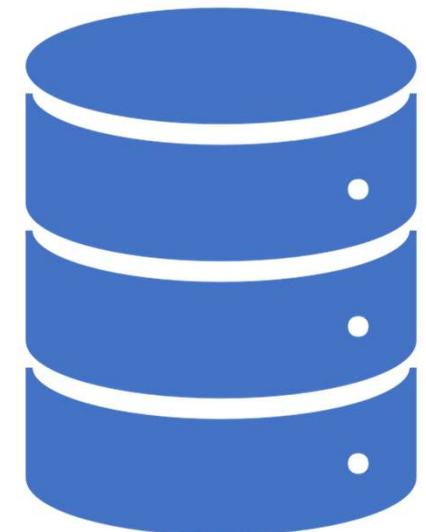
Queue name: vrm:queue  
Queue type: TRANSIENT (Default)  
Send correlation id: AUTO (Default)

OK Cancel



# Connector : Database

- Mandatory config details required
  - Driver
  - Host
  - Port
  - User
  - Password
  - Database/service name/ instance
  - SQL Query text



# Notes to Remember about Database

In Mule3, we used to have one database connector and we will be selecting the kind of operation that we want. Like Select, Update, Insert & Delete

In Mule4, We have separate connector for each operation

You can test the connection if you want to check if connection is successful

The “Output” type of select statement result is in “Array”

# SQL Configuration

The screenshot displays the Mule Studio interface with several components:

- test-saikiran-connectorsFlow:** A flow diagram showing a "Listener" (represented by a globe icon) connected to a "Select" step (represented by a database icon). The "Select" step is also connected back to the "Listener".
- Global Element Properties - Database Config:** A configuration dialog for a database connector. It shows a "Name" field set to "Database\_Config" and a "Connection" dropdown set to "MySQL Connection". The "General" tab is selected, showing fields for "Host" (localhost), "Port" (8081), "User" (root), "Password" (redacted), and "Database" (mule). A note indicates the "MySQL JDBC Driver" is missing and needs to be added.
- Message Flow:** A configuration panel for the "Select" step. It shows the connector configuration is set to "Database\_Config". The "Query" section contains the SQL query: "Select \* from employees".
- connectorsFlow:** A flow diagram showing three steps: "Insert", "Update", and "Delete", each represented by a database icon.

# Notes to Remember about Database

FTP and SFTP are one and same except that SFTP used secured protocol

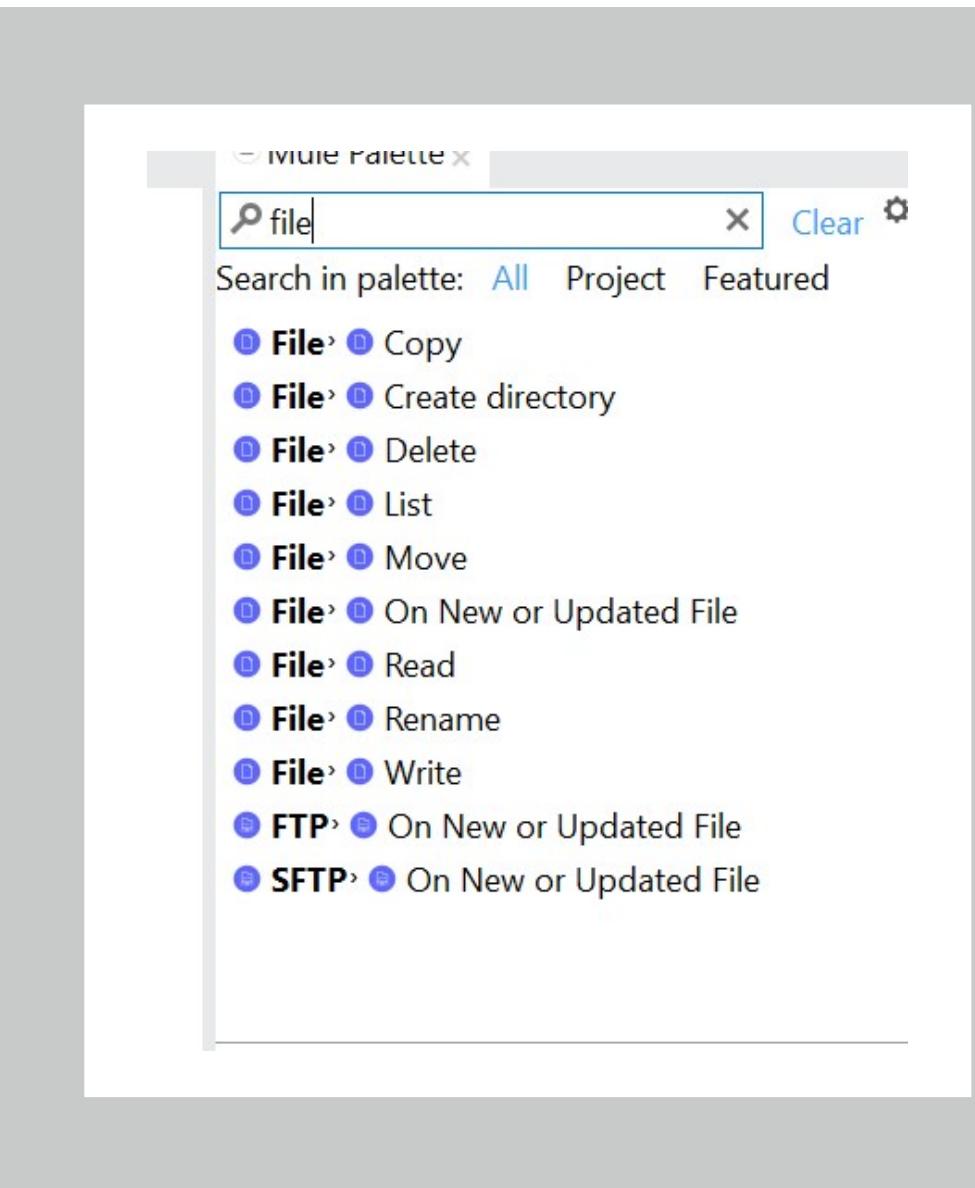
The way we connect the operations are all same for SFTP and FTP

Requires credentials for SFTP

Host, Port, username & Password are Mandatory configuration properties

# Connector: File

- Mandatory Configuration details
  - There are no particular configuration details to configure a file as it does not have any username & password
  - But the configuration need to be created for sure
- In Mule4, We have separate connectors for each operation & You can check the connection whether it is successful
- By Default, If we read a file, The file will not be deleted
- You can extract the file name, size etc using attributes



# Email connector

- Email connector is used to send emails through smtp protocol
- Dummy smtp address
  - <https://pepipost.com/>
  - Sign up the above url and got to sandbox to use dummy smtp server

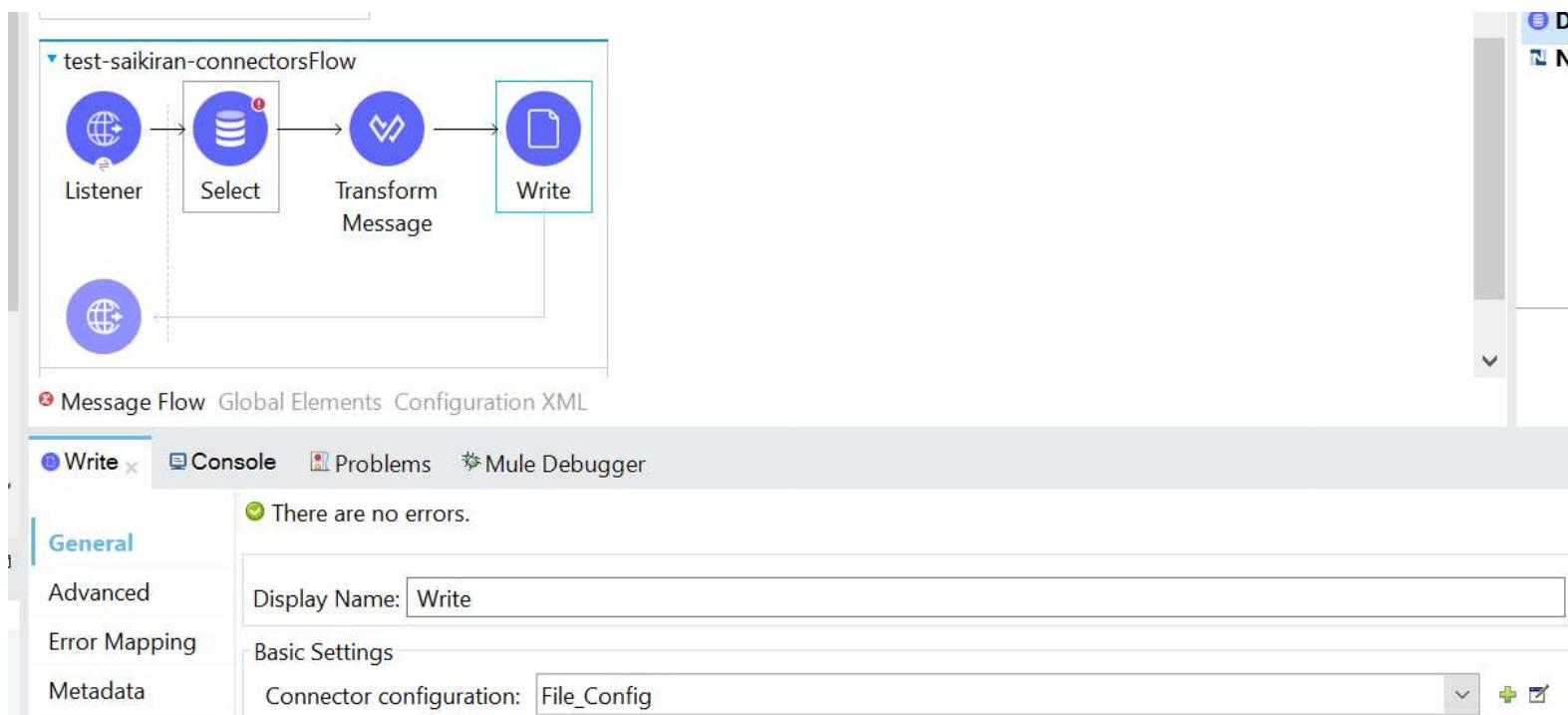


# Some Important operations in FTP/SFTP

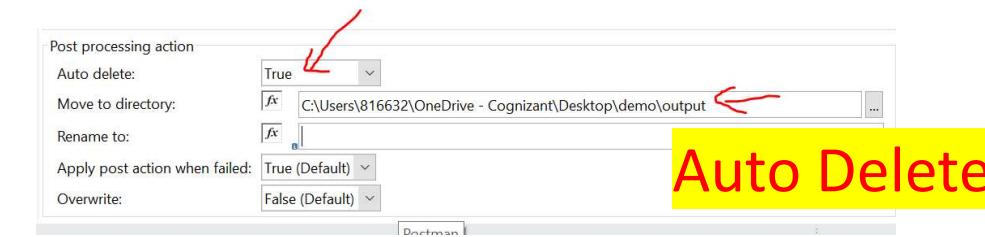
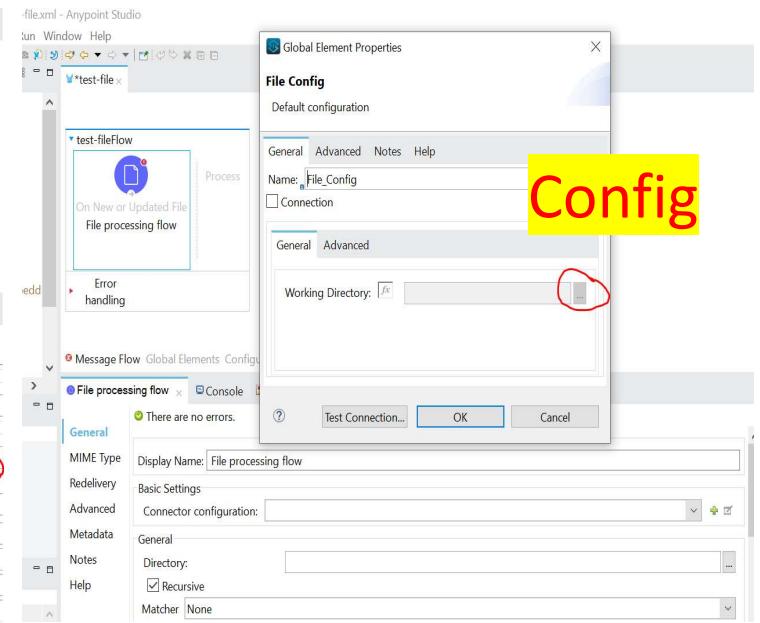
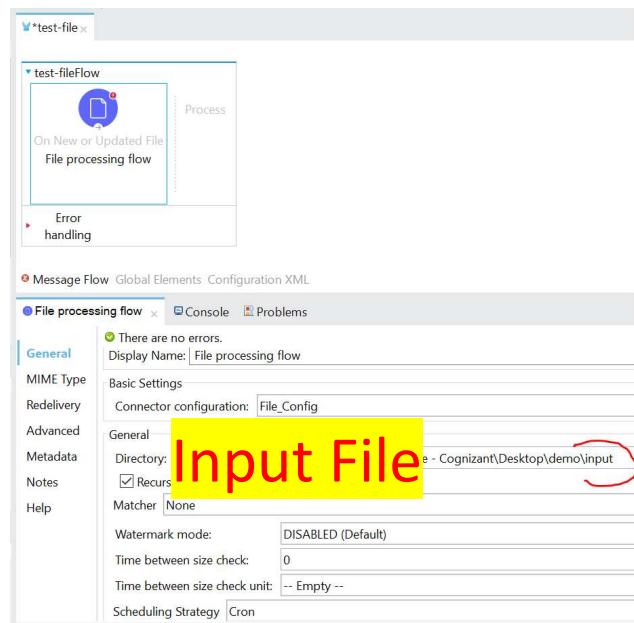
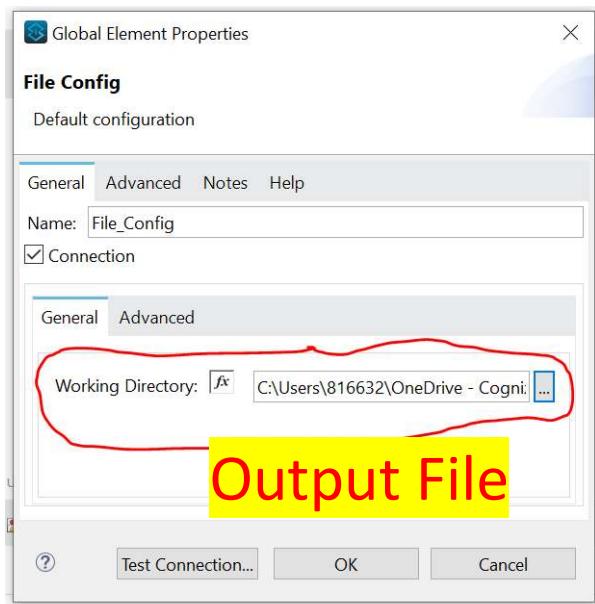


# File:Write Operation

It will write/Store what ever data coming from Database to a File

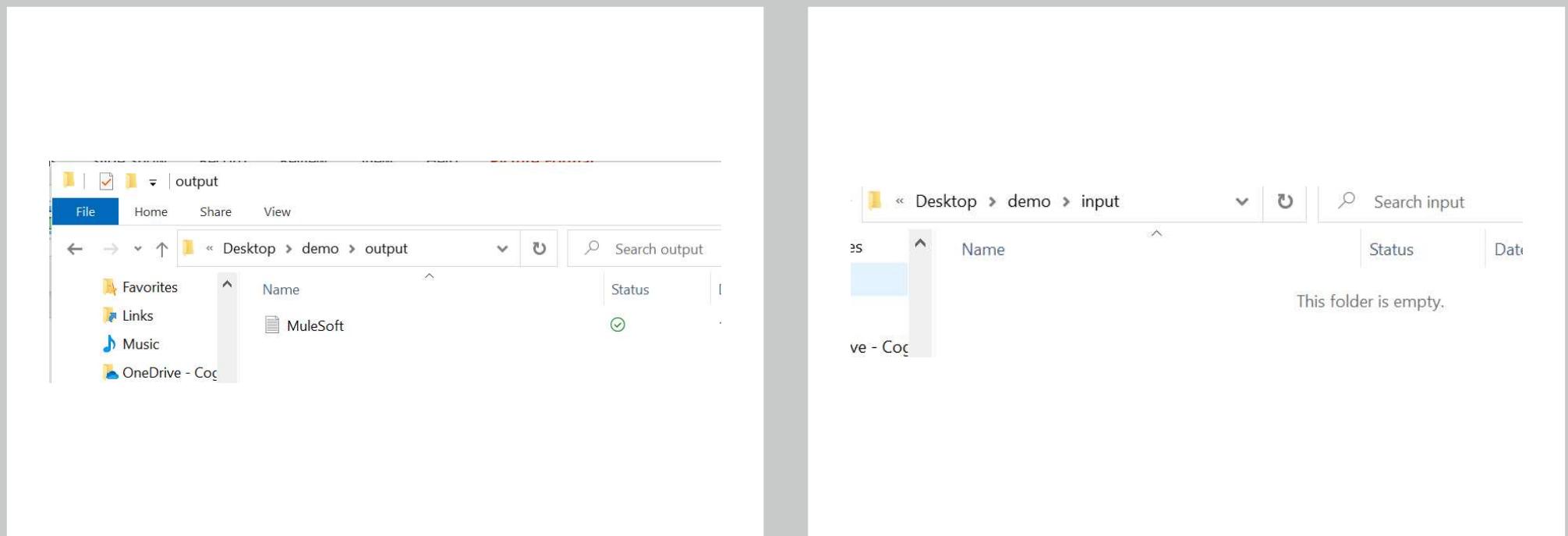


# Trigger a flow when new file is created/Modified



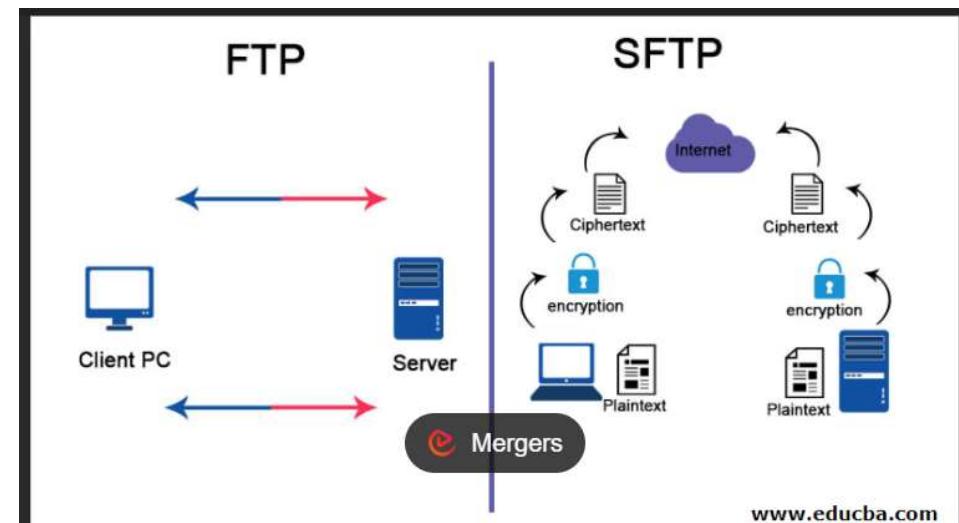
# Output

- Run the flow
- Place a demo file in Input
- After Time Frequency it will move to the output folder & file will be deleted from input folder



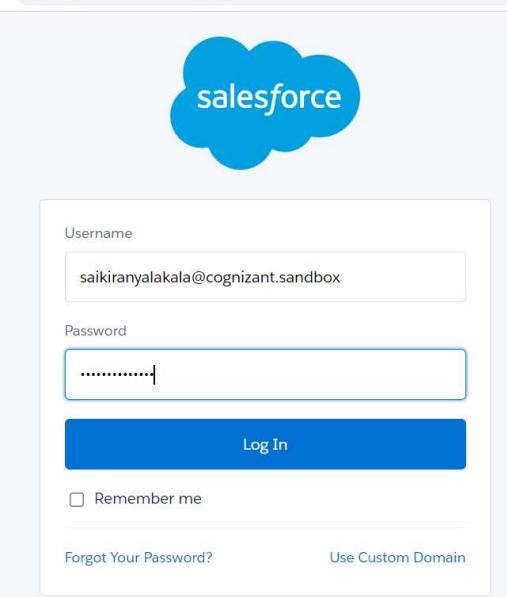
# Difference between File/FTP/SFTP

File	FTP	SFTP
Used to connect local system	Used to connect remote system	Used to connect remote system securely
Not requires any setup	Winscp is required software	Winscp, rebex sftp server softwares are required

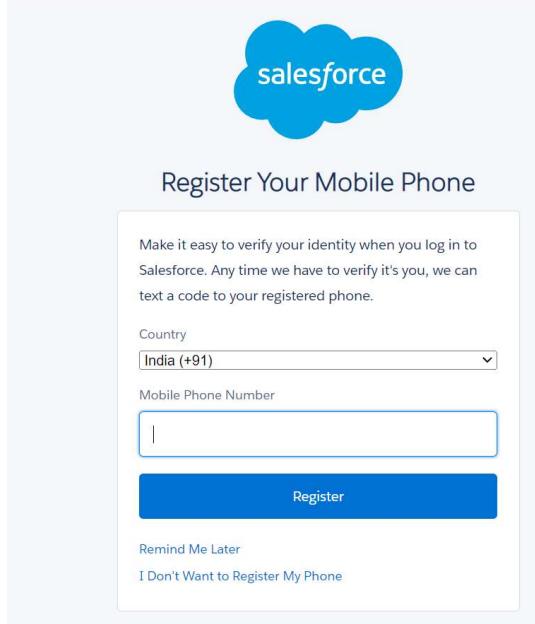


## Connector: Salesforce

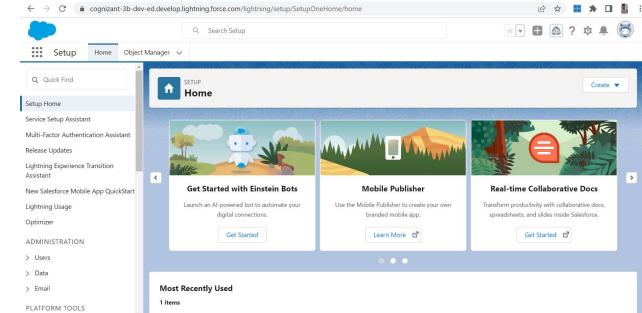
- Create Salesforce Account : <https://developer.salesforce.com/>



The screenshot shows the Salesforce login interface. It features a large blue cloud logo at the top. Below it is a form with fields for 'Username' (containing 'saikiranyalakala@cognizant.sandbox') and 'Password' (with masked input). A 'Log In' button is prominently displayed in a blue box at the bottom. At the very bottom of the page are links for 'Forgot Your Password?' and 'Use Custom Domain'.



This screenshot shows the 'Register Your Mobile Phone' page. It includes a large blue cloud logo. The main text explains the purpose of phone verification. Below this are fields for 'Country' (set to 'India (+91)') and 'Mobile Phone Number'. A 'Register' button is at the bottom. There are also 'Remind Me Later' and 'I Don't Want To Register My Phone' links.



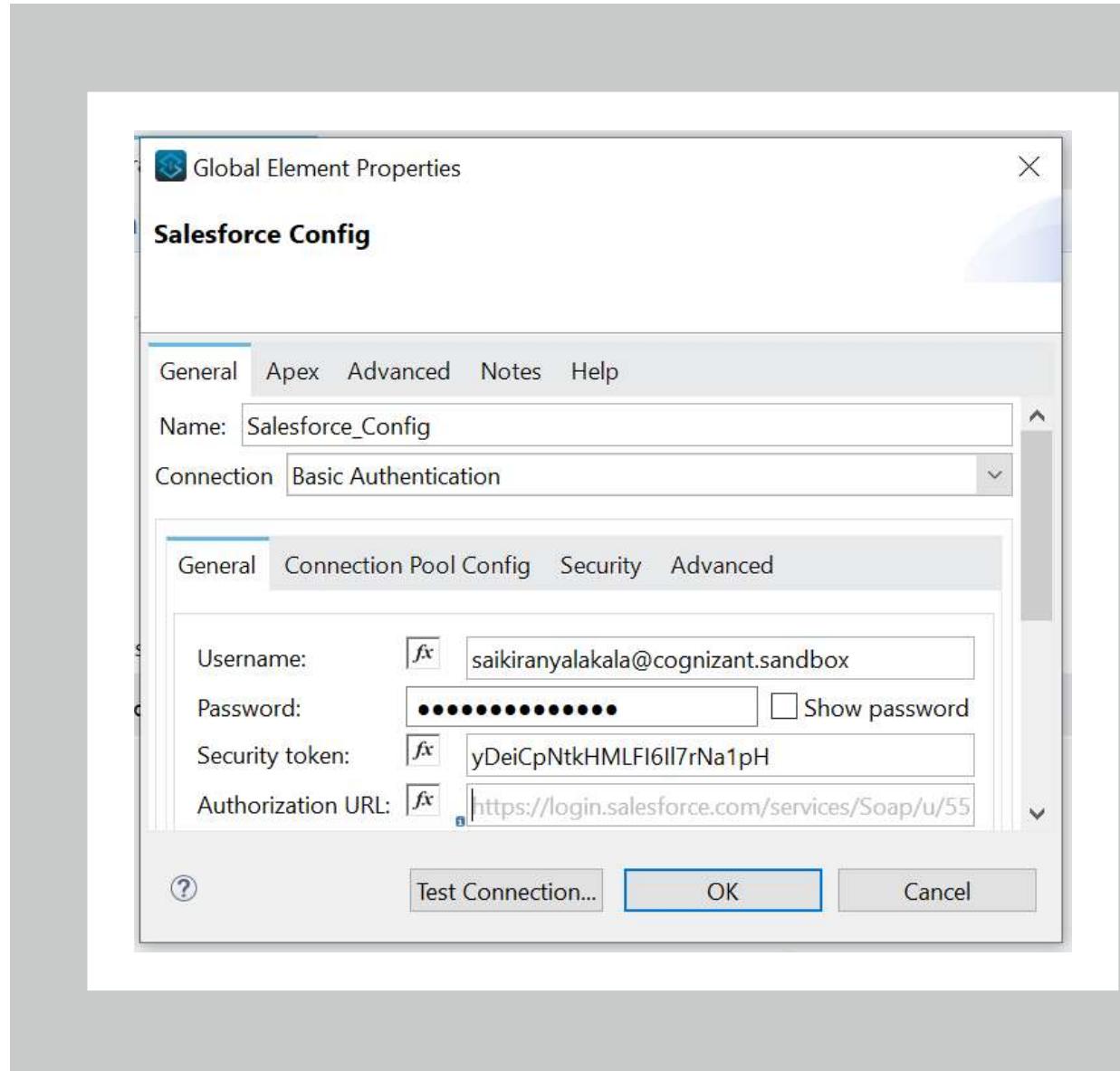
The screenshot displays the Salesforce Setup Home page. The URL in the browser bar is 'cognizant-3b-dev-ed.develop.lightning.force.com/lightning/setup/SetupOneHome/home'. The page has a sidebar with 'Quick Find' and links like 'Setup Home', 'Service Setup Assistant', etc. The main content area features three cards: 'Get Started with Einstein Bots', 'Mobile Publisher', and 'Real-time Collaborative Docs'. A 'Most Recently Used' section is also visible.

Security token is Important to connect  
-> *Click on reset security token to receive it on mail*

The screenshot shows the Salesforce Setup interface. The top navigation bar includes a cloud icon, a search bar labeled "Search Setup", and various navigation icons. The main menu on the left is titled "Setup" and includes sections like "My Personal Information", "Advanced User Details", "Approver Settings", "Authentication Settings for External Systems", "Change My Password", "Connections", "Grant Account Login Access", "Language & Time Zone", and "Login History". The current page, "Reset My Security Token", is displayed under the "Object Manager" section. The page title is "Reset Security Token". A descriptive text explains that a security token is needed for logging in from untrusted IP addresses or desktop clients. A note states that after resetting, the old token cannot be used. A "Reset Security Token" button is visible at the bottom of the page.

# Salesforce Config

- Mandatory Config details required:
  - Username
  - Password
  - Security token
  - Auth url
- In Mule4, We have separate connectors for each operation & You can check the connection whether it is successful



## Important points about salesforce

- Salesforce accepts **application/java** (i.e, Array of objects)

http://localhost:8081/test

```
1  {
2  ...
3  ...
4  ...
5  ...
6  ...
7  }
```

Message Flow Diagram:

```
graph LR; Listener((Listener)) --> Transform[Transform Message]; Transform --> Create((Create));
```

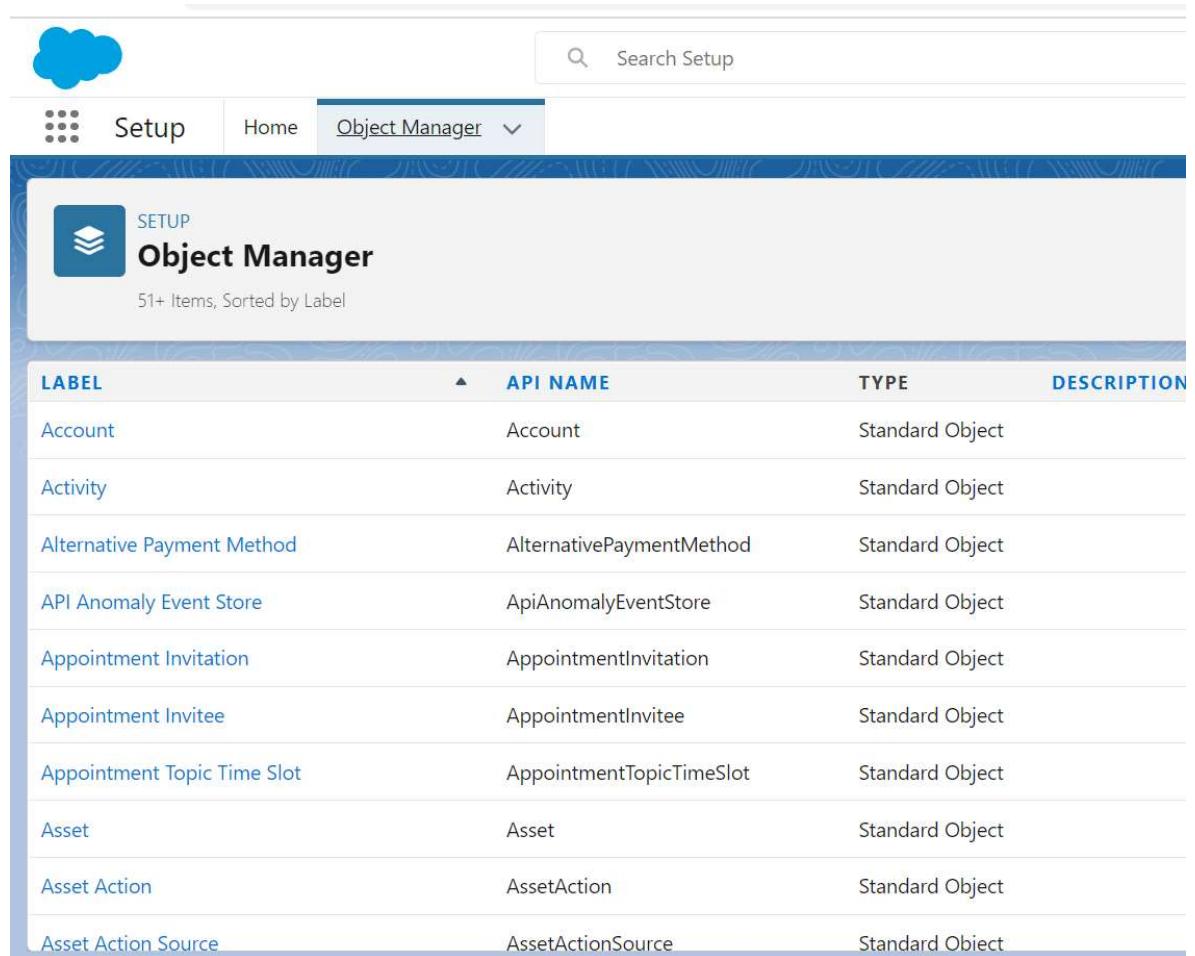
Transform Message Configuration:

- Input: Payload : Any [Define metadata](#)
- Variables: Variables
- Attributes: Attributes : Any [Define metadata](#)
- Output Payload: 1. %dw 2.0  
2. output application/java  
3. ---  
4. payload

Create Component Configuration (highlighted with red circles):

- Display Name: Create
- Connector configuration: Salesforce\_Config
- Type: Account
- Records: 1 payload

# Go to Object Manager



The screenshot shows the Salesforce Setup interface with the following details:

- Header:** Includes the Salesforce logo, a search bar labeled "Search Setup", and navigation tabs for "Setup", "Home", and "Object Manager".
- Section Header:** "SETUP Object Manager" with a subtitle "51+ Items, Sorted by Label".
- Table:** A list of objects with the following columns: LABEL, API NAME, TYPE, and DESCRIPTION.
- Data:** The table lists 11 standard objects:

LABEL	API NAME	TYPE	DESCRIPTION
Account	Account	Standard Object	
Activity	Activity	Standard Object	
Alternative Payment Method	AlternativePaymentMethod	Standard Object	
API Anomaly Event Store	ApiAnomalyEventStore	Standard Object	
Appointment Invitation	AppointmentInvitation	Standard Object	
Appointment Invitee	AppointmentInvitee	Standard Object	
Appointment Topic Time Slot	AppointmentTopicTimeSlot	Standard Object	
Asset	Asset	Standard Object	
Asset Action	AssetAction	Standard Object	
Asset Action Source	AssetActionSource	Standard Object	

# Look into the Fields closely and Apply the Fields present in Postman

The screenshot shows the Salesforce Object Manager interface for the 'Account' object. The top navigation bar includes 'Setup', 'Home', and 'Object Manager'. The left sidebar lists various setup options: Details, Fields & Relationships (which is selected), Page Layouts, Lightning Record Pages, Buttons, Links, and Actions, Compact Layouts, Field Sets, Object Limits, Record Types, and Related Lookup Filters. The main content area is titled 'Fields & Relationships' and displays a table of 40 items, sorted by Field Label. The table columns are: FIELD LABEL, FIELD NAME, DATA TYPE, CONTROLLING FIELD, and INDEXED. The data in the table is as follows:

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Account Name	Name	Name		✓
Account Number	AccountNumber	Text(40)		
Account Owner	OwnerId	Lookup(User)		✓
Account Site	Site	Text(80)		
Account Source	AccountSource	Picklist		▼
Active	Active_c	Picklist		▼
Annual Revenue	AnnualRevenue	Currency(18, 0)		
Billina Address	BillinaAddress	Address		

Account  
created in  
Salesforce

GET http://localhost:8081/test

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

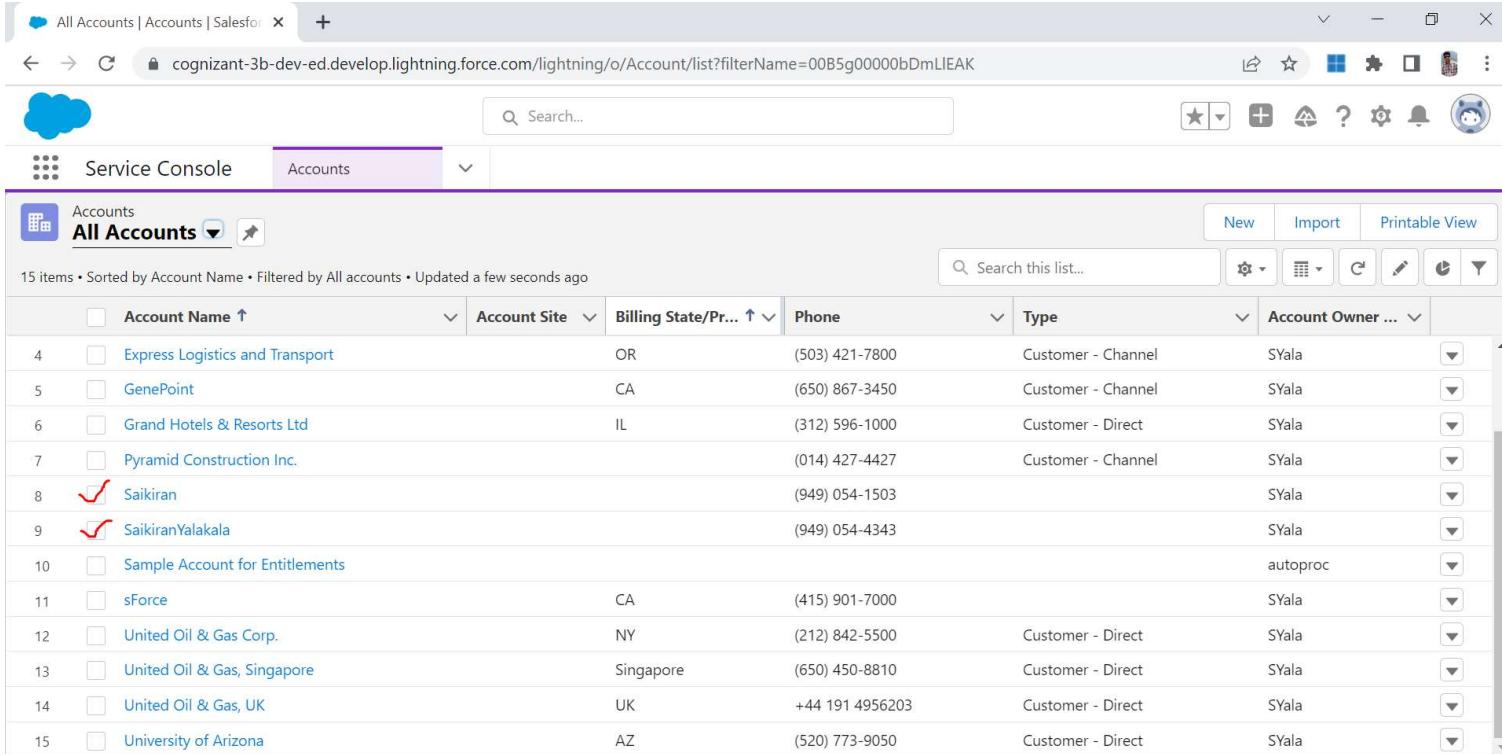
```
1 [  
2   {  
3     "Name": "SaikiranYalakala",  
4     "Phone": "9490544343",  
5     "BillingCity": "Bangalore"  
6   }]
```

Body Cookies Headers (3) Test Results 200 OK 16.99 s 45

Pretty Raw Preview Visualize JSON

```
1 {  
2   "id": null,  
3   "items": [  
4     {  
5       "exception": null,  
6       "message": null,  
7       "payload": {  
8         "success": true,
```

# Created Accounts in Salesforce Accounts



The screenshot shows the Salesforce Lightning Experience interface. The top navigation bar includes the Salesforce logo, a search bar, and various icons. Below the header, the Service Console tab is selected, followed by the Accounts tab. The main content area displays a list of accounts under the heading "All Accounts". The list includes columns for Account Name, Account Site, Billing State/Prov, Phone, Type, and Account Owner. The account "SaikiranYalakala" is highlighted with red checkmarks in the first and second columns. The account "Saikiran" is also highlighted with a red checkmark in the first column.

	Account Name	Account Site	Billing State/Prov	Phone	Type	Account Owner
4	Express Logistics and Transport		OR	(503) 421-7800	Customer - Channel	SYala
5	GenePoint		CA	(650) 867-3450	Customer - Channel	SYala
6	Grand Hotels & Resorts Ltd		IL	(312) 596-1000	Customer - Direct	SYala
7	Pyramid Construction Inc.			(014) 427-4427	Customer - Channel	SYala
8	Saikiran			(949) 054-1503		SYala
9	SaikiranYalakala			(949) 054-4343		SYala
10	Sample Account for Entitlements					autopro
11	sForce	CA		(415) 901-7000		SYala
12	United Oil & Gas Corp.		NY	(212) 842-5500	Customer - Direct	SYala
13	United Oil & Gas, Singapore		Singapore	(650) 450-8810	Customer - Direct	SYala
14	United Oil & Gas, UK		UK	+44 191 4956203	Customer - Direct	SYala
15	University of Arizona		AZ	(520) 773-9050	Customer - Direct	SYala

# Queries in Salesforce

---

Select -> Used for select records from the database

---

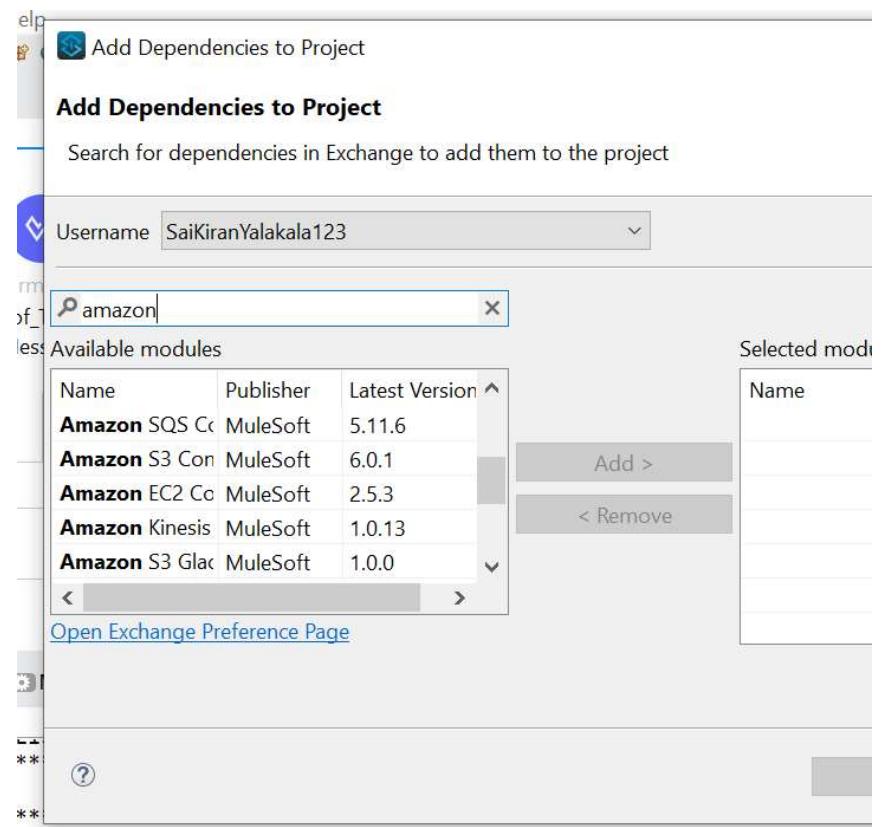
Upsert -> Used for inserting and updating the records

---

Delete -> Used to delete the records

# Connector : AWS-S3

- Mandatory Config details
  - Access key
  - Secret key
  - Region
- Where can we get Access key and Secret key
  - Inside Profile -> Security Credentials -> Access Key -> Displayed Over there (or) Create New
- Aws cannot be found in modules, Install it from Exchange
- We will call files as Bucket ins AWS



# Error handling

- What is an error?
  - An error occurs when an unexpected event happens while processing
- What Is Error handling?
  - Exception handling is nothing but handling the errors while unexpected event happens while processing
- Levels of error handling:
  - Flow level error handling
  - Connector level error handling
  - Custom error handling
  - Global error handling

Core ➔  Error Handler

Core ➔  On Error Continue

Core ➔  On Error Propagate

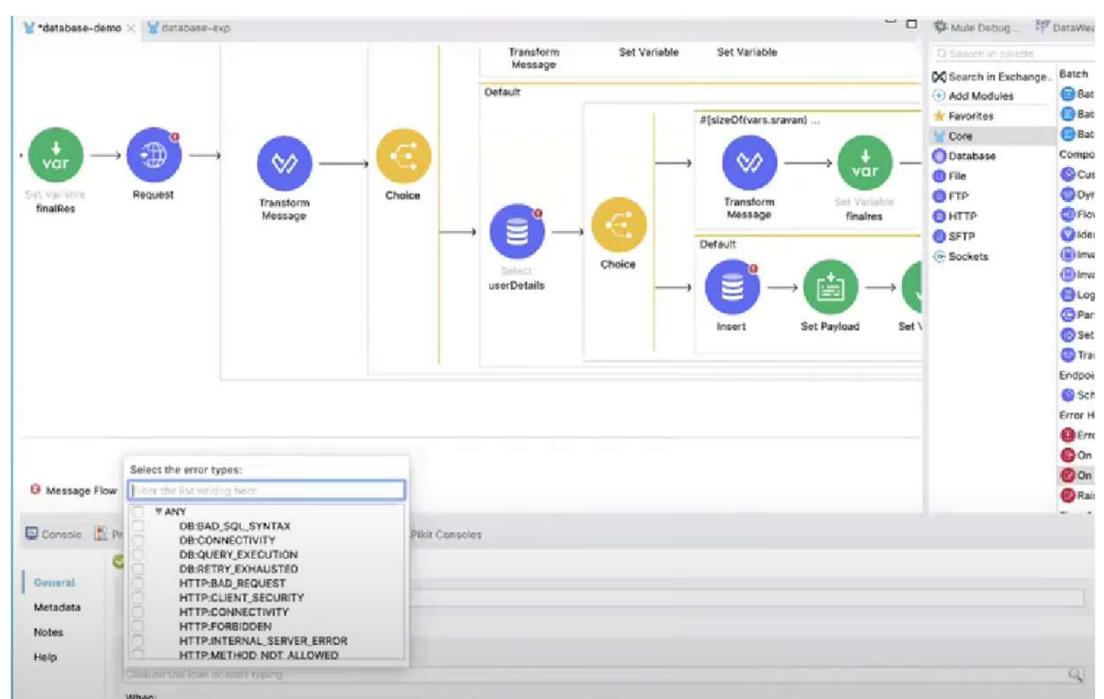
Core ➔  Raise error

# At what levels we can do Error handling ?

- At project level using **default error handler**
  - If we are not using any error handler, By default MuleSoft will handle the error and gives proper error message to the user
- At project level using **custom global error handler**
- Mule4 error handling is very easy when compared with Mule3
- At **flow level** in exception handling using
  - On error continue
  - On error propagate
  - Raise error
- With in flow or at process level using **Try scope**

# Error handling : Important Notes

- When even an **error** occurs in the flow, An **error Object** is Created
- It contains many properties (error.description, error.errorType, etc., )
- ErrorType is combination of Name space and Identifier
  - Eg: **HTTP:UNAUTHORIZED**
- Sub flow does not have any error handling
- Either it is propagate or continue, Mule executes all the components with in the Block
- How to identify the errors
  - Mule4 have the excellent feature on identifying the errors, By connectors we can able to identify the errors

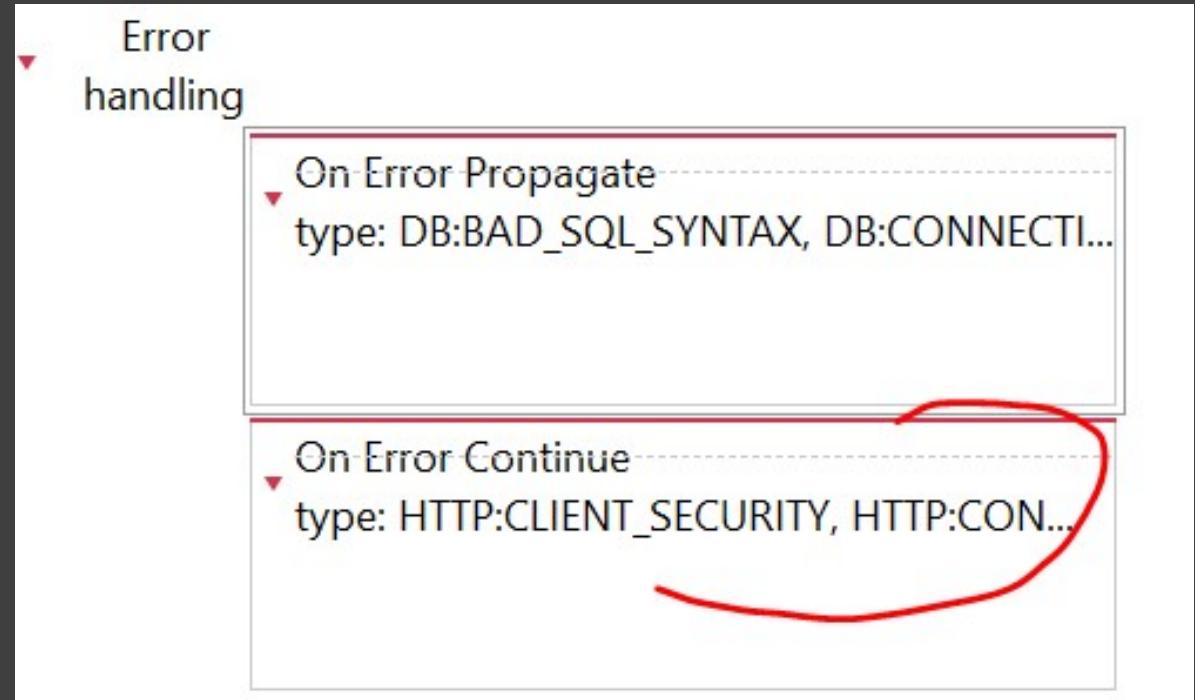
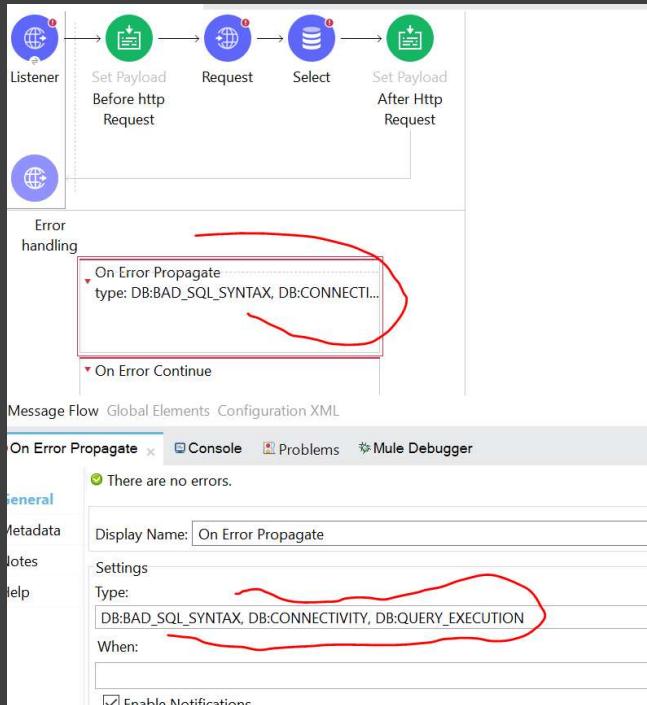


# Rules to understand Error handling

---

- See if anything is present in error handling
- Even if there are on-error propagate or continueblocks, see If that particular errorType is handled
- If Not, Mule will use default error handling
- If flow is called by any other flow, then it will raise the error to calling flow





Placing DB errors in  
Error propagate & HTTP  
errors in Error Continue

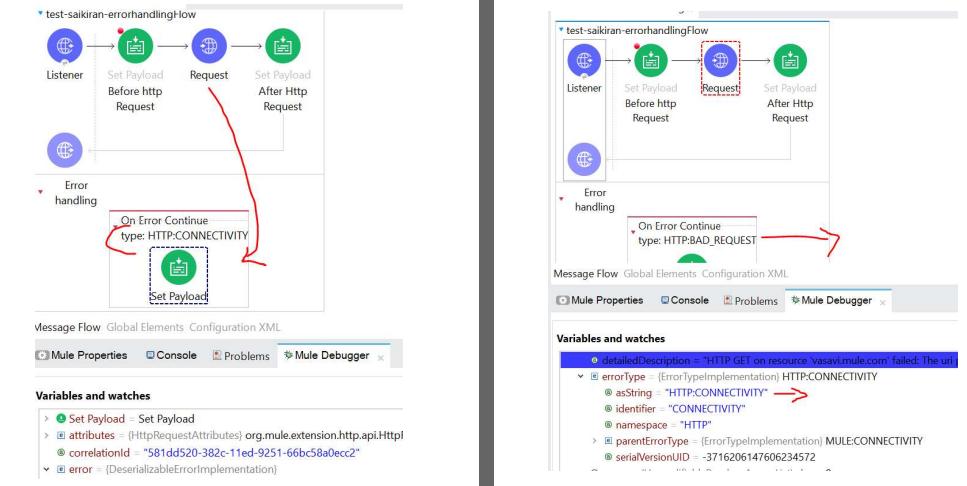
- ON **ERROR CONTINUE** scope always returns a success response to the next level
- ON **ERROR PROPAGATE** always propagates the **error** to the next level and returns an **error** response to the next level.

# Creating Dummy HTTP request to see How the Error are handled by Default Error handler

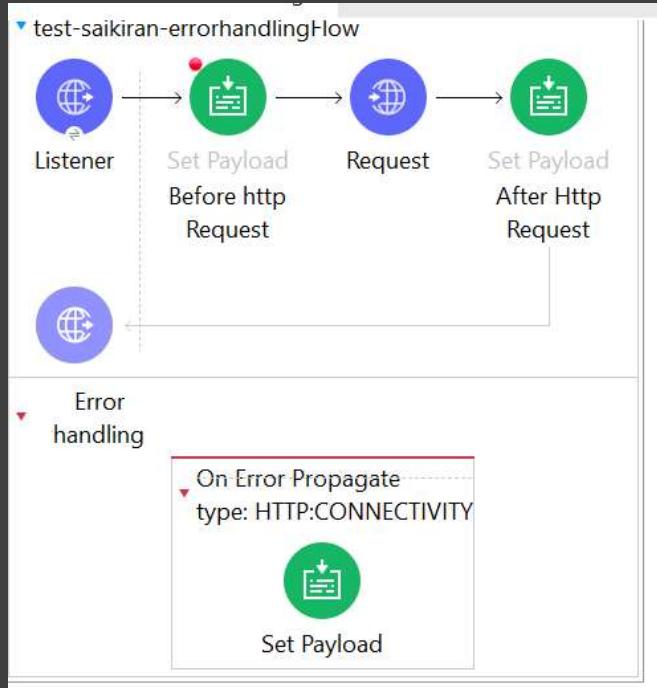
The screenshot shows the Mule Studio interface with two main panes. The top pane displays a message flow named 'test-saikiran-errorhandlingFlow'. The flow starts with a 'Listener' component, followed by a 'Set Payload Before http Request' component, a 'Request' component (which is highlighted with a red dashed box), and a 'Set Payload After Http Request' component. The bottom pane shows the 'Request' configuration details. The 'Request' tab is selected, displaying fields for 'Display Name' (Request), 'Basic Settings' (Configuration: 'HTTP\_Request\_configuration', URL: 'http://localhost:8081'), and 'Request' (Method: 'GET (Default)', Path: '/', URL: 'vasavi.mule.com'). A red arrow points from the 'URL' field to the word 'DUMMY!' written above it. The bottom right pane is a 'Mule Debugger' window showing a variable tree. A red checkmark is placed next to the 'description' field under the 'error' node, which contains the error message: 'HTTP GET on resource 'vasavi.mule.com' failed: The uri provided 'vasavi.mule.com' must contain a scheme.' Another red checkmark is placed next to the 'asString' field under the 'errorType' node, which contains the value 'HTTP:CONNECTIVITY'.

# On Error Continue

- On placing **HTTP:CONNECTIVITY** error in the error handling block it will handle the error and triggers 200 response to with the payload message
- On placing **HTTP:OTHERS** error in the error handling block it cannot handle the error and 500 response will be sent with the error message



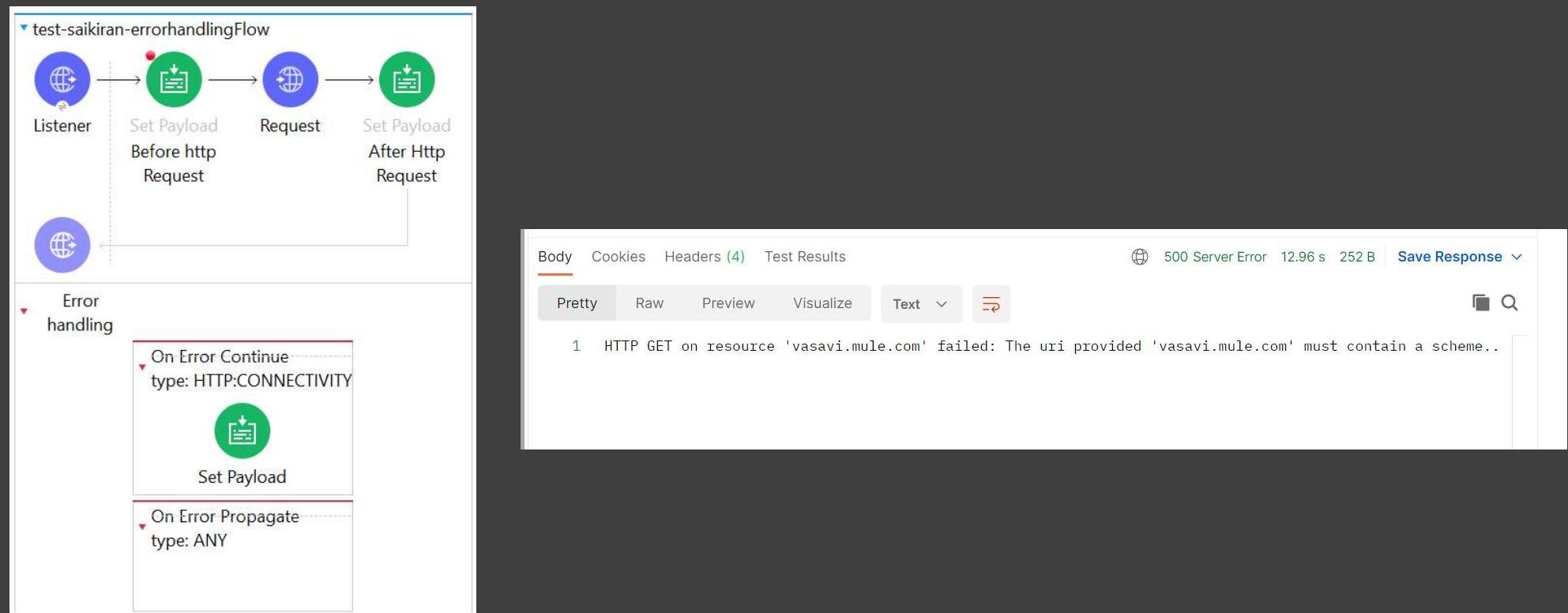
The screenshot shows a POSTMAN interface with a GET request to `http://localhost:8081/error`. The 'Body' tab is selected, displaying the response: `1 Connectivity error handled`. The status bar at the bottom right indicates a successful response with code `200 OK`, time `1 m`, and duration `51.18 s`.



## On Error Propagate

- On placing **HTTP:CONNECTIVITY** error in the error handling block it will handle the error and triggers 500 response
- It is going to the calling flow if there is any error to handle with in On-Error Propagate





What happens if we Place On-error continue & Propagate as above

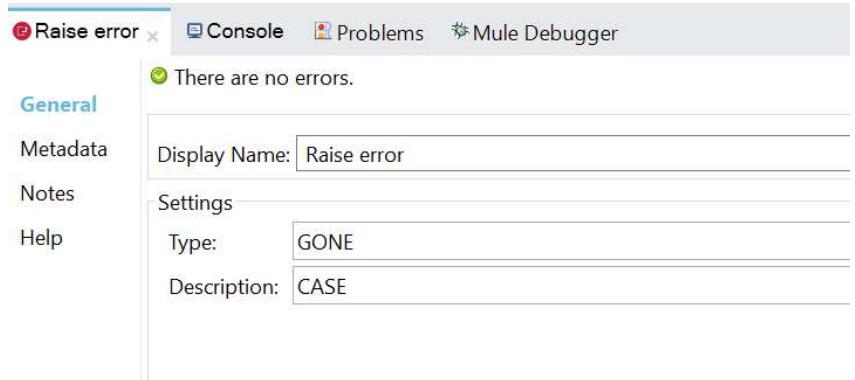
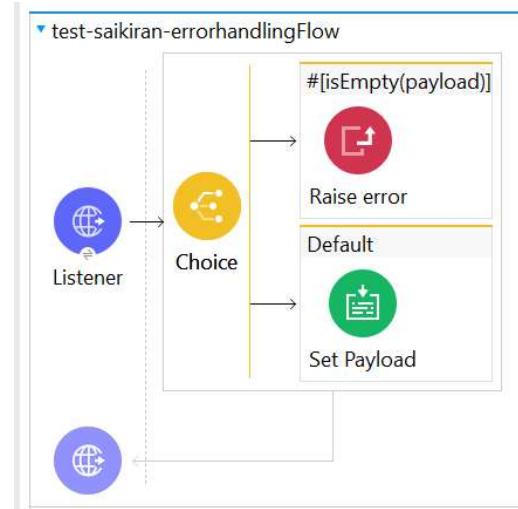
- Placing a dummy URL in the request (HTTP:CONNECTIVITY)
- In below case Propagate will execute and send 500 status to calling flow

# On Error Propagate vs On Error Continue

On Error Propagate	On Error continue
<b>ON ERROR PROPAGATE</b> always propagates the <b>error</b> to the next level and returns an <b>error</b> response to the next level	<b>ON ERROR CONTINUE</b> scope always returns a success response to the next level
If the error is handled by using the On-Error Propagate, It will raise an error back to the calling flow	On error continue will not raise the error back to the calling flow and continue to next processor after “Flow-Ref” and continue further process as it is
Capture error and stop the execution of the flow and returns the error code with 500.. Status code	Captures the error and continues to the next level for execution, It always returns 200 status code

# Raise Error:

- We can have own type and Own description in Raise Error
- It's not like On-error (or) Continue to have default error cases
  - Eg: HTTP:NOT ALLOWED, DB:CONNECTIVITY
- We cannot use the default error cases like HTTP, DB etc..., It should be defined by us



# Scopes in Mulesoft

Try

Cache

First  
successful

Until  
successful

Async

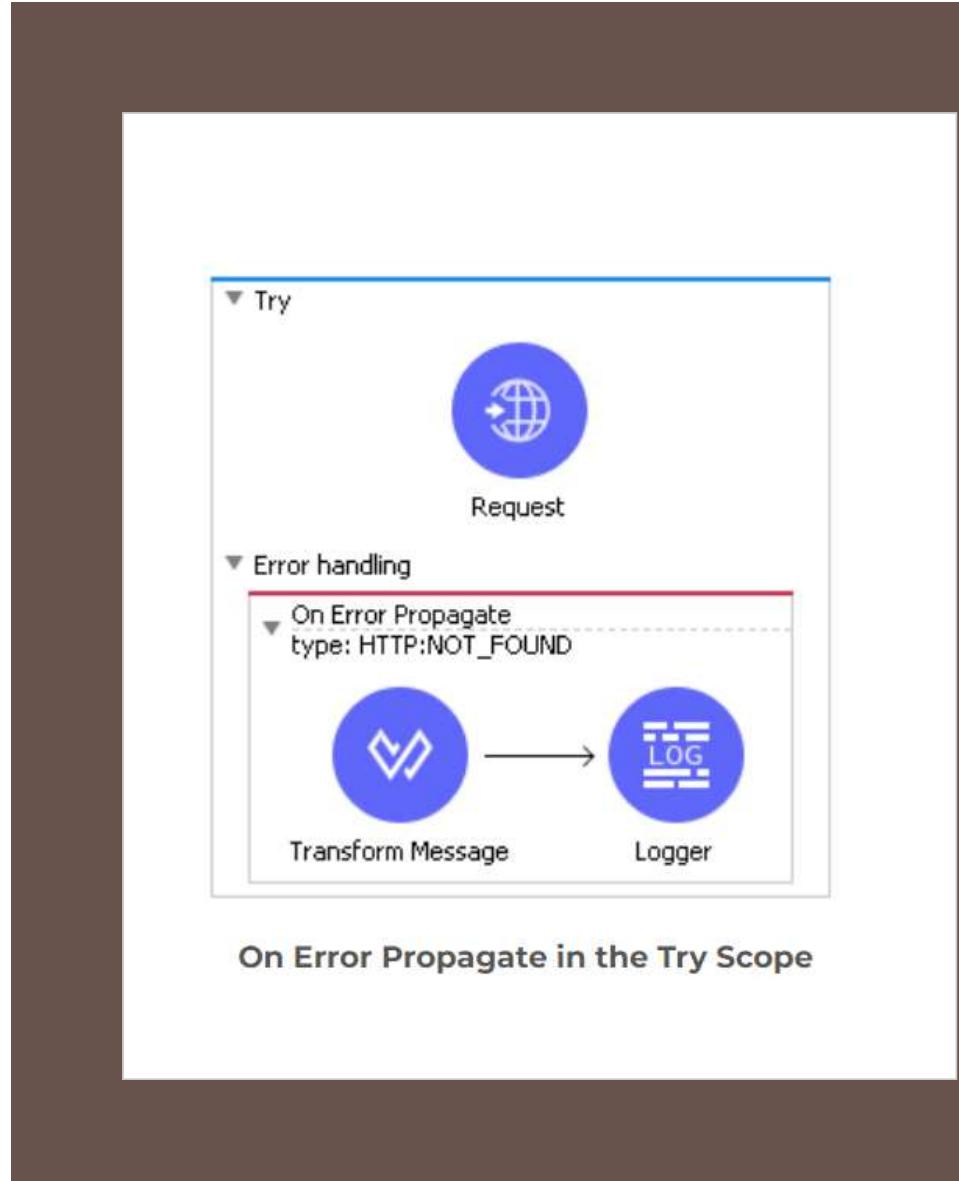
For each  
scope

Message  
enricher

# Try scope

---

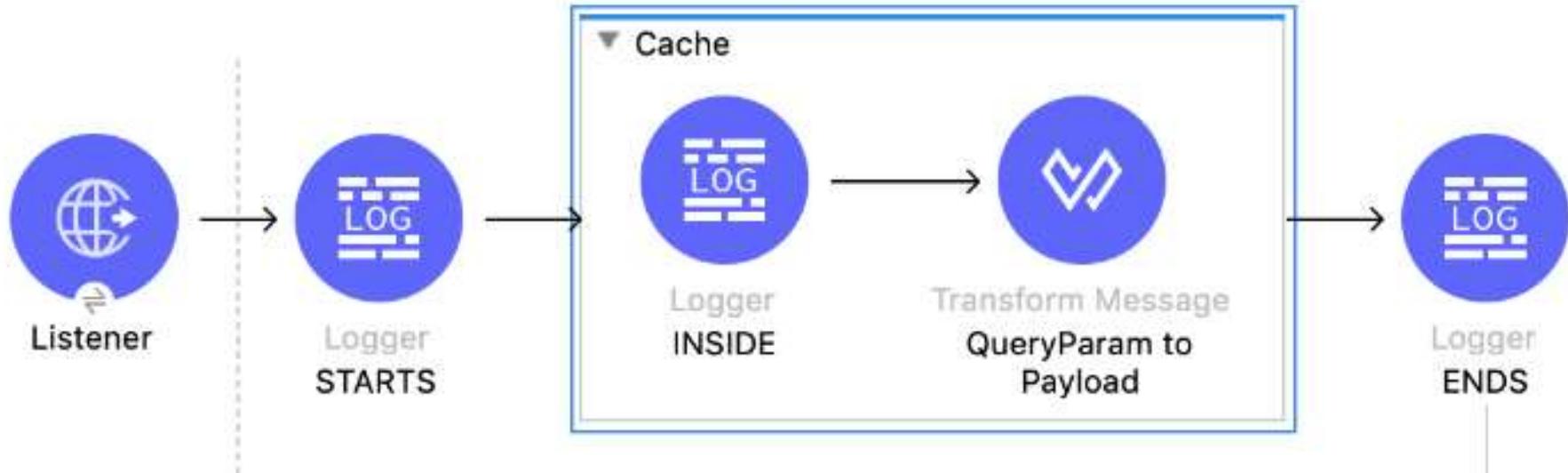
Try scope is used to handle the errors within a connector level



## Cache scope

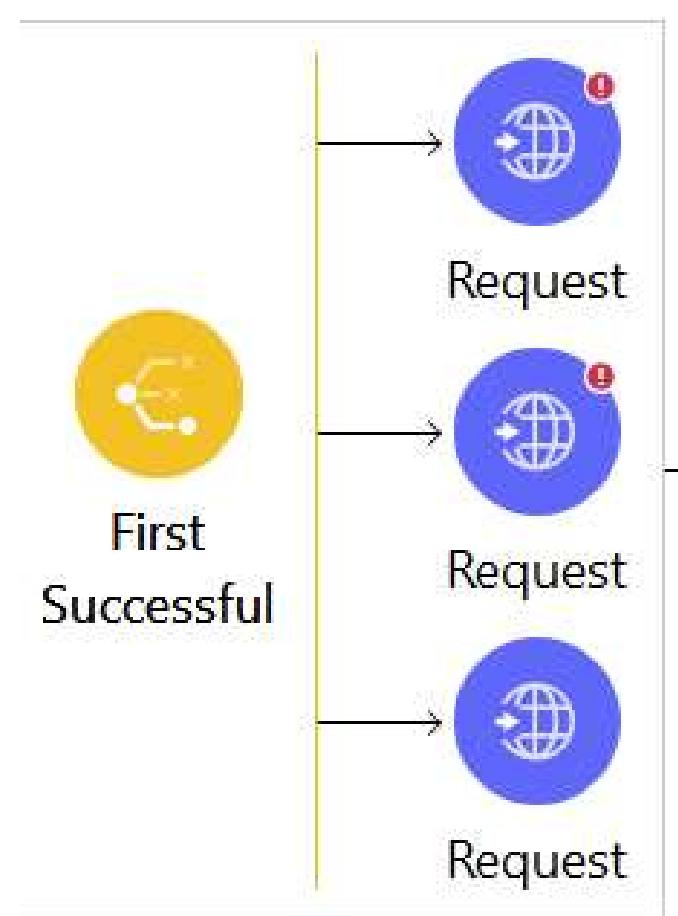
- Cache scope will store previous requests and responses.
- If there is any similar request, We will retrieve information from memory(Object store) instead of calling target system

### ▼ OSv2CacheOnMule4



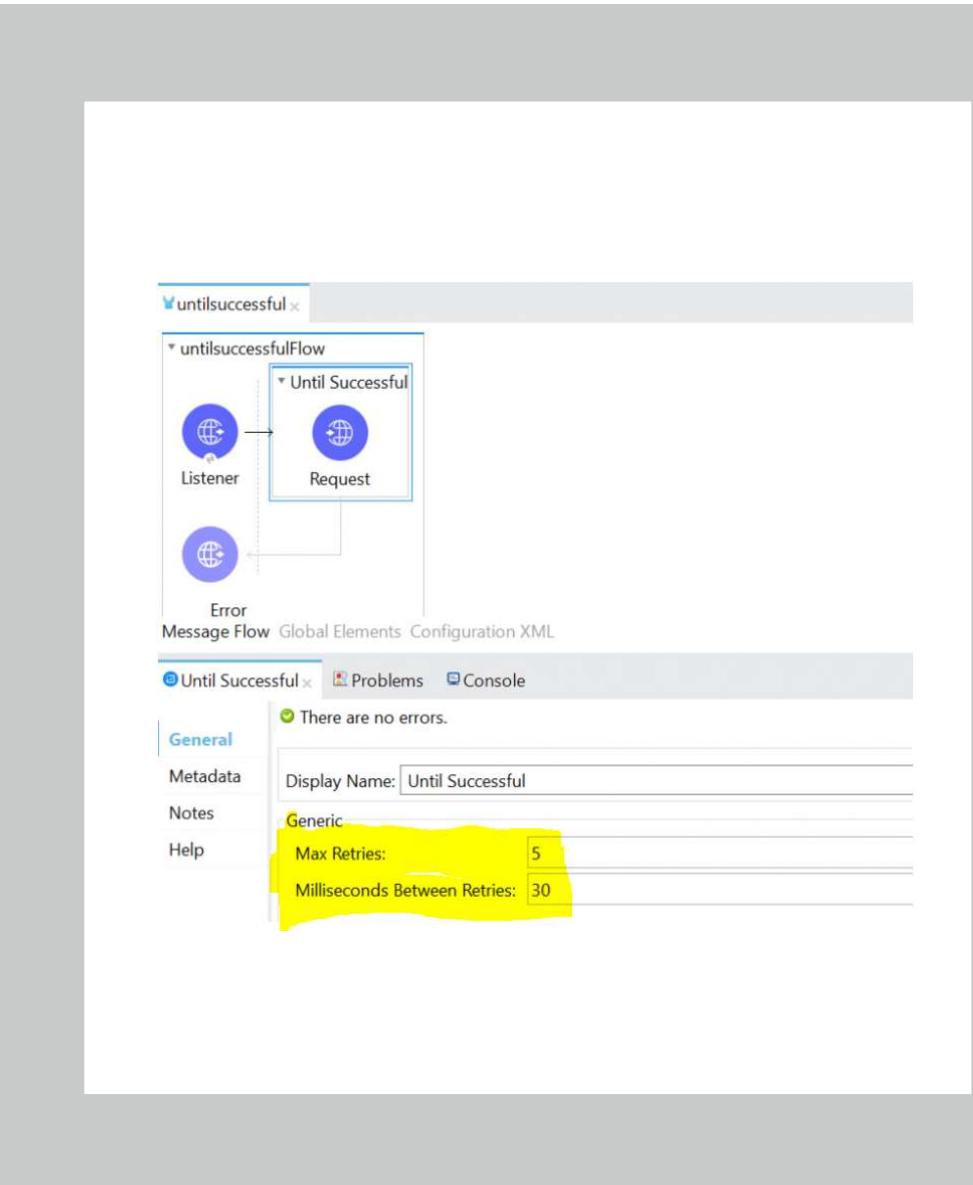
# First Successfull

- The First Successful router iterates through a list of configured processing routes until one of the routes executes successfully. If any processing route fails execution (throws an error), the router executes the next configured route.
- It is also similar to the scatter gather but in case of scatter gather parallel processing is done from a single source system to multiple subscriber systems but in case of first successful it executes the first successful record and ignores the remaining



# Until successful

- The Until Successful scope processes the components within it until they succeed or exhaust the maximum number of retries. Until Successful runs synchronously.
- **Max Retries:** Specifies the maximum number of retries that are attempted
- **Milliseconds Between Retries:** Specifies the minimum interval between two attempts to process, in milliseconds, Default value is 1 Min

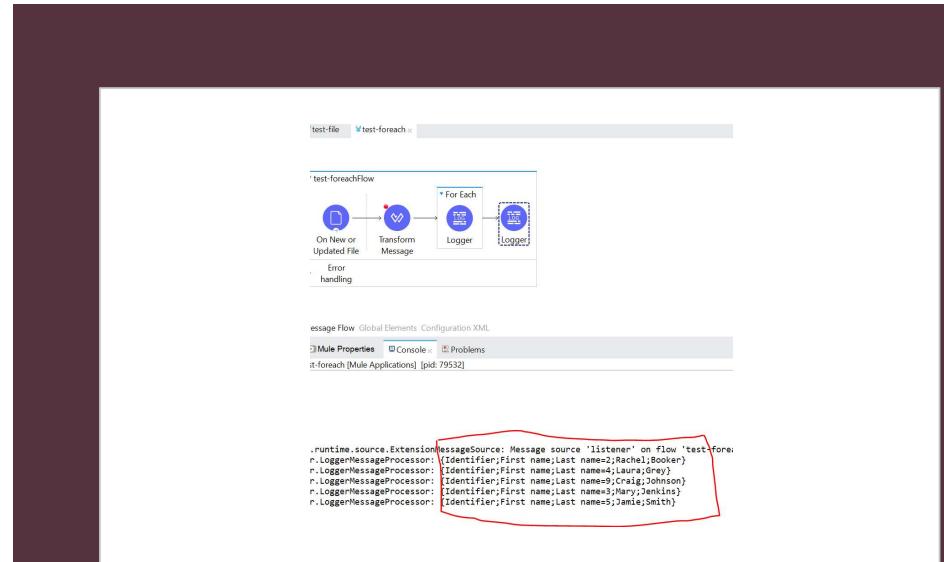


# FOR EACH

1. For each is used to process the records synchronously(1 by 1). It is a single threaded operation
2. Sample CSV files:

<https://support.spatialkey.com/spatialkey-sample-csv-data/>  
<https://support.staffbase.com/hc/en-us/articles/360007108391-CSV-File-Examples>

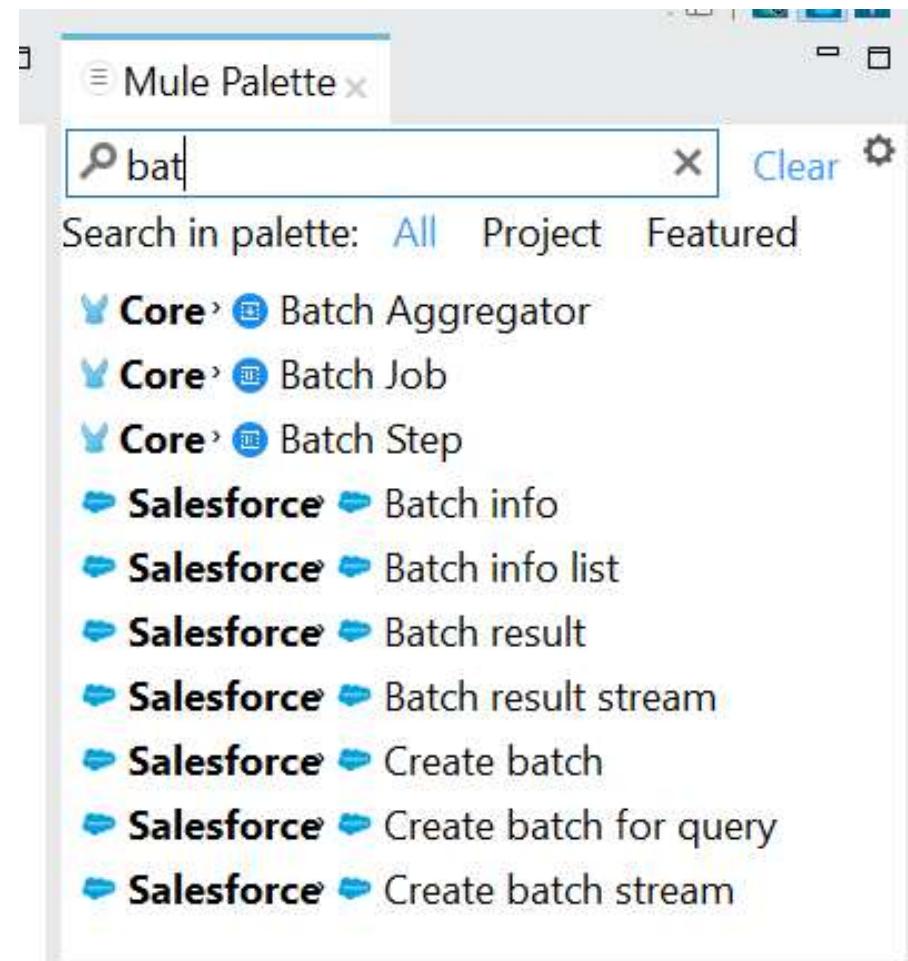
3. POC: Take (On new or Update) file connector -> take input as CSV file record -> Transform it to Java -> Print records 1by1 in logger(For each scope)



	A	B	C	D
1	Identifier;First name;Last name			
2	2;Rachel;Booker			
3	4;Laura;Grey			
4	9;Craig;Johnson			
5	3;Mary;Jenkins			
6	5;Jamie;Smith			
7				
8				
9				
10				

# Batch Processing

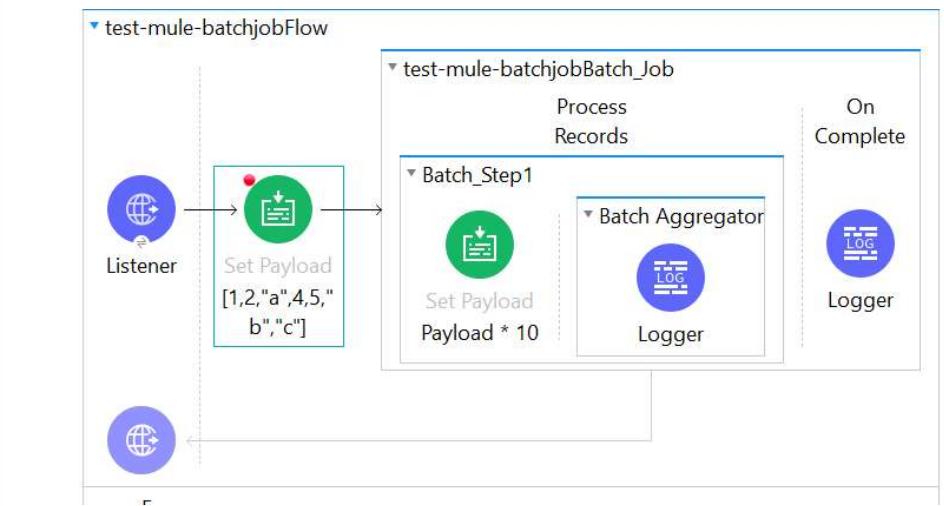
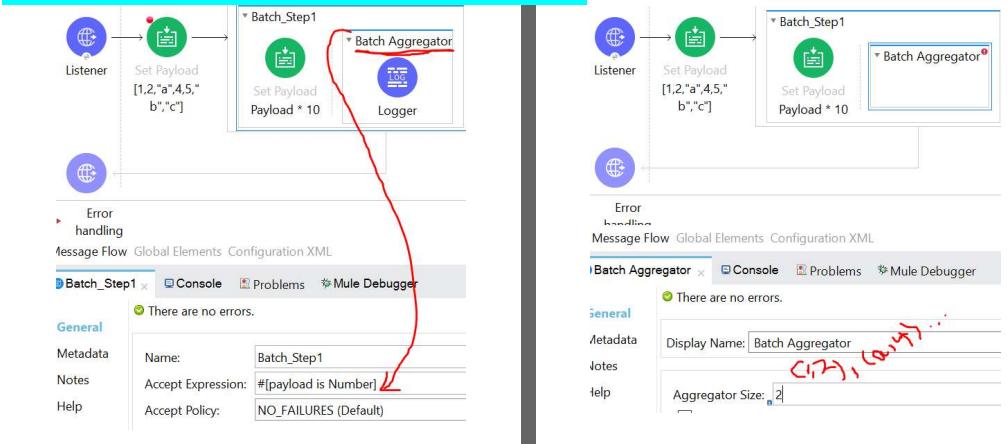
- The messages are processed in batch
- In Mule3 we have 3 phases in Batch job
  - Input phase
  - Process phase
  - On complete phase
- Input phase is removed in Mule4, We have only 2 phases
  - Load and Dispatch phase
  - Process phase
  - On complete phase



# Real Time : Batch processing

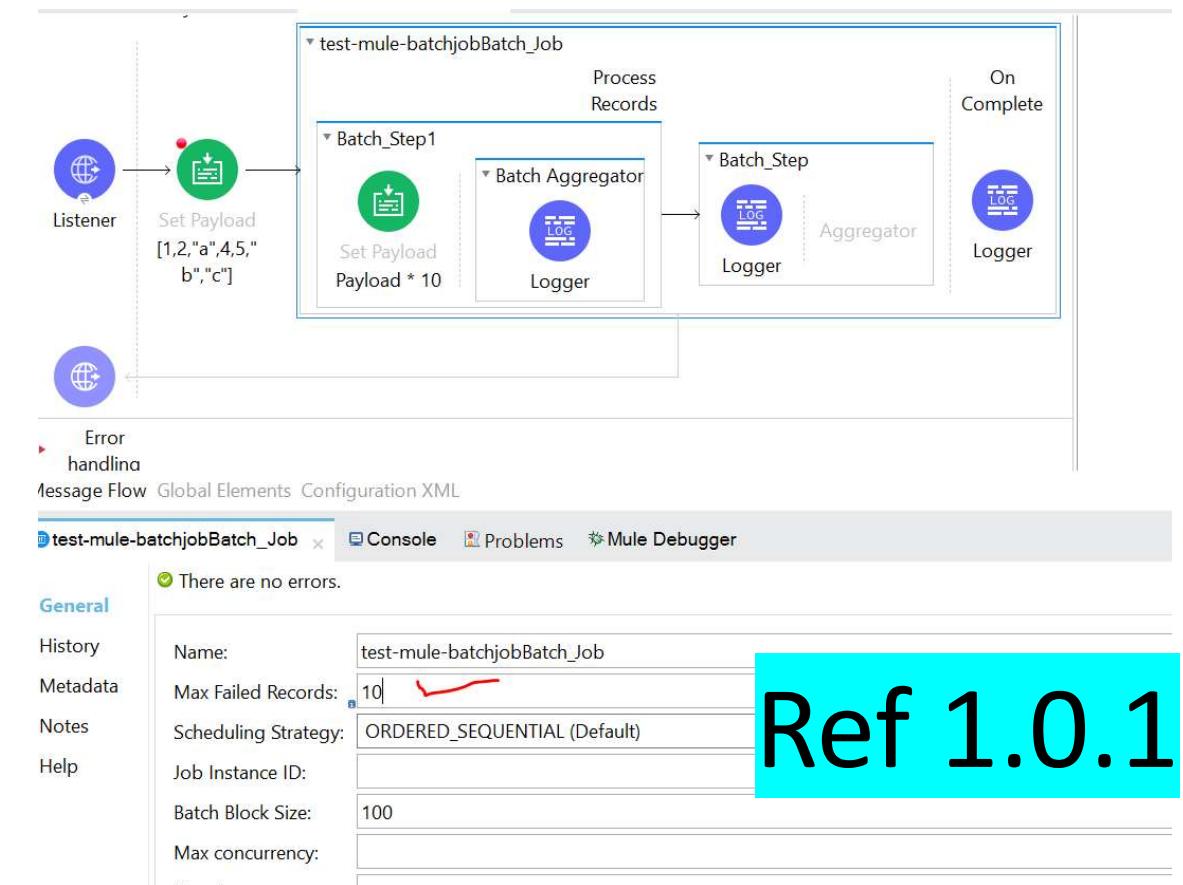
- Take an array of inputs in the Payload (Eg: [1,2,"a",4,5,"b","c"])
- Drag the batch job
- We can add 'n' number **batch steps** in **Process records**
- **Processor:** If there is any Logic like inserting and upserting the DB we will use processor
- **Aggregator:** The number of processes inside data to be aggregated with

## Process only numbers



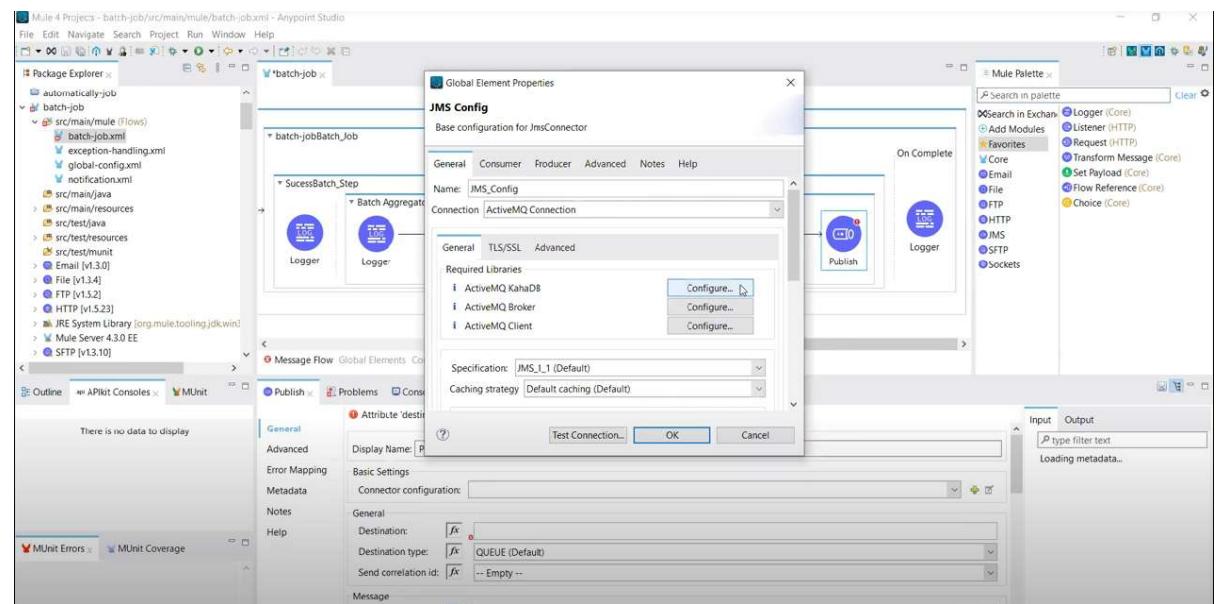
## In 2<sup>nd</sup> Batch step Mentioning only failures(Strings)

- Note: In Batch\_JOB, Keep Max failed records as some number, Otherwise failed records will not be triggered to next Batch\_step. **Ref:1.0.1**



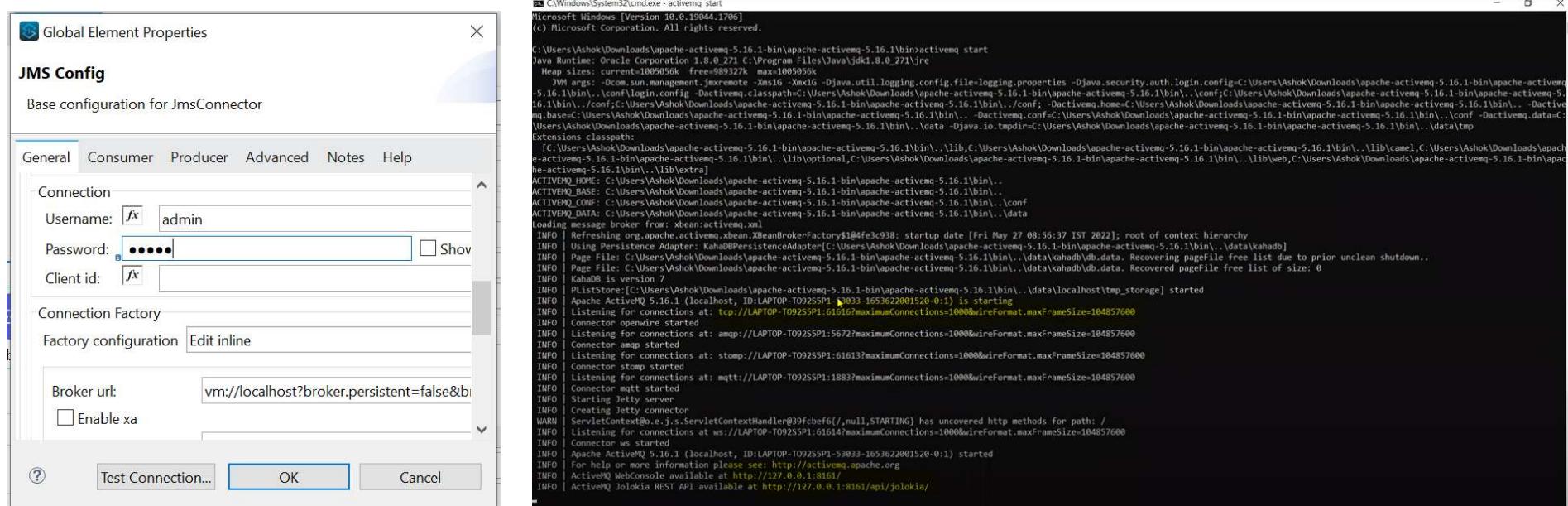
# ActiveMQ

- Apache **ActiveMQ** is an open source message broker written in Java together with a full Java Message Service (JMS) client
- If there are any records failing in Batch processing we can put them in queue and reprocess it again
- Configure JMS config in MQ
- We need to install tomcat Apache ActiveMQ  
<https://activemq.apache.org/>

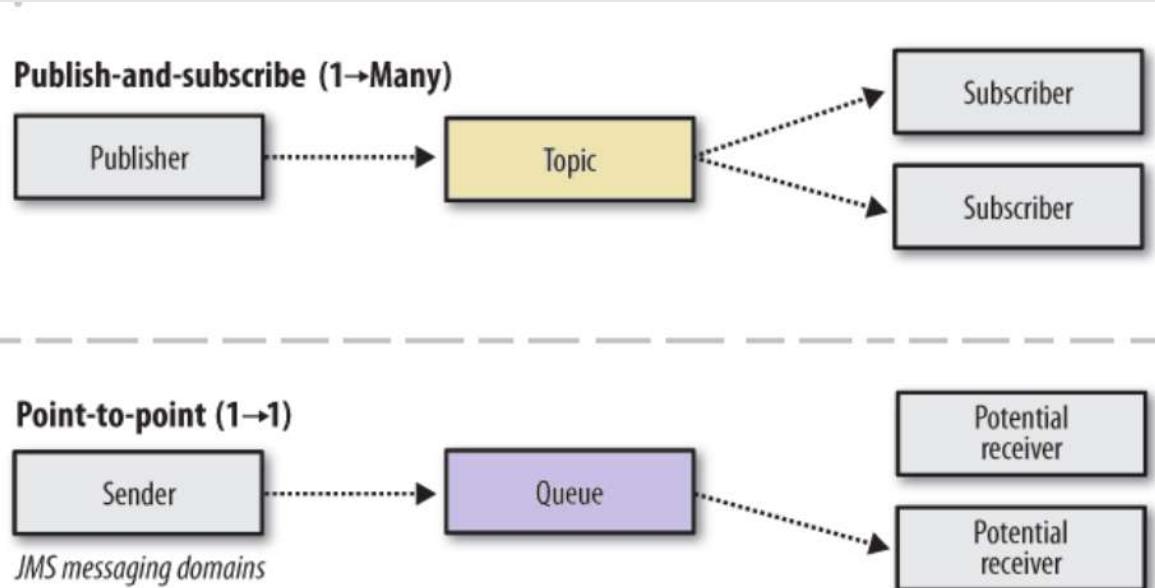


# To start ActiveMQ server

- In Studio connector config Always username and Password should be admin
  - In apache activemq download, Go to bin path-> enter cmd -> press **activemq start** to start the server



# Difference between Queue and Topic



- Point to Point integration is known as Queue (1-1)
- Publisher subscriber model is known as Topic (1-n)
- A queue can have only one consumer, whereas a topic can have multiple subscribers.

# ActiveMQ dashboard

- We have publish to publish the message to the queue
- Consume is used to consume the published queue in another or same queue
- publishConsume is used to publish and consume in the same queue

The screenshot shows two parts of the ActiveMQ Admin interface. The top part is a configuration dialog for a queue named 'Publish'. It includes fields for 'Display Name' (set to 'Publish'), 'Connector configuration' (set to 'JMS\_Config'), and 'Destination' (set to 'productitems'). The 'productitems' entry is highlighted with a yellow box. The bottom part is a list of queues, showing one queue named 'productitems' with 0 pending messages and 0 consumers. The 'productitems' entry is also highlighted with a yellow box.

There are no errors.

Display Name: Publish

Basic Settings

Connector configuration: JMS\_Config

General

Destination:  productitems

Destination type:  QUEUE (Default)

Send correlation id:  -- Empty --

Message

localhost:8161/admin/queues.jsp

Support | Logout

The Apache Software Foundation  
http://www.apache.org/

ActiveMQ™

Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send

Queues:

Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
ashok	0	0	0	0	Browse Active Consumers Active Producers <input type="button" value="View"/> <input type="button" value="Edit"/>	Send To Purge Delete Pause
productitems	0	0	0	0	Browse Active Consumers Active Producers <input type="button" value="View"/> <input type="button" value="Edit"/>	Send To Purge Delete Pause

Copyright 2005-2020 The Apache Software Foundation.

■ Queue Views  
▪ Graph  
▪ XML

■ Topic Views  
▪ XML

■ Subscribers Views  
▪ XML

■ Useful Links  
▪ Documentation  
▪ FAQ  
▪ Downloads  
▪ Forums

## MUnits in Mule4

Munit is a developer level mock testing done by the developer after complete development process and the below are the operations to perform Munit testing

Search in Exchange...

Add Modules

Favorites

Core

Database

HTTP

MUnit

MUnit Tools

Salesforce

Sockets

VM

Operations

Set Event

Set null payload

Scopes And Routers

After Suite

After Test

Before Suite

Before Test

Test

Search in Exchange...

Add Modules

Favorites

Core

Database

HTTP

MUnit

MUnit Tools

Salesforce

Sockets

VM

Operations

Assert that

Fail

Mock when

Run custom

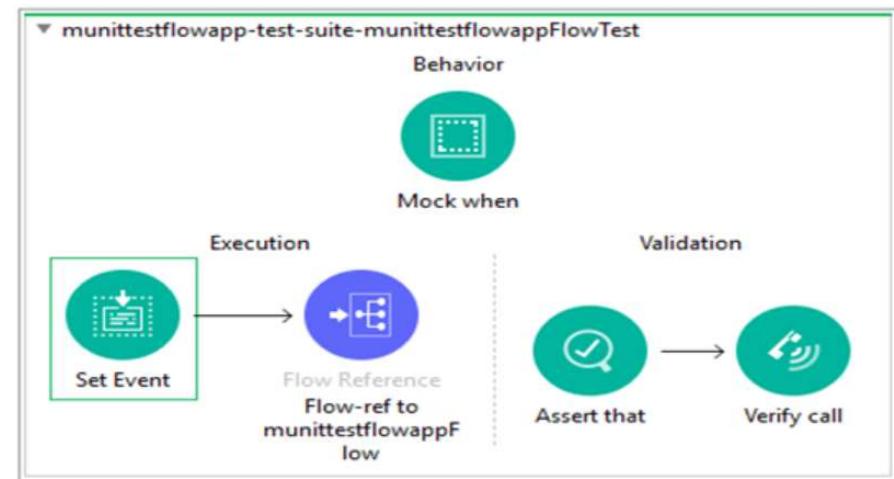
Verify call

Scopes And Routers

Spy

# Munits Recorder & Manual testing

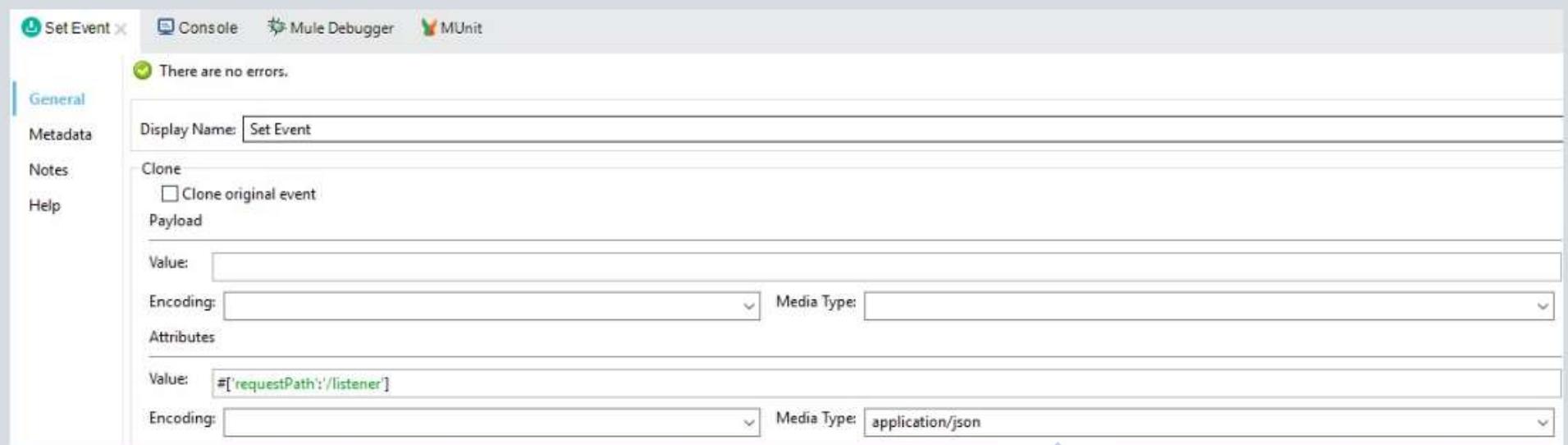
- Munit recorder is used to generate the test validation automatically by postman call. It is an automated testing done by developer but pin Manual testing every setup need to be done by developer manually



## Set Event

At the beginning of a Munit test, this processor can be used to indicate that the first message is sent to the flow for the purpose of testing. The set-event has the cloneOriginalEvent property. If set true, then it clones the event produced by the code and by default this property is false.

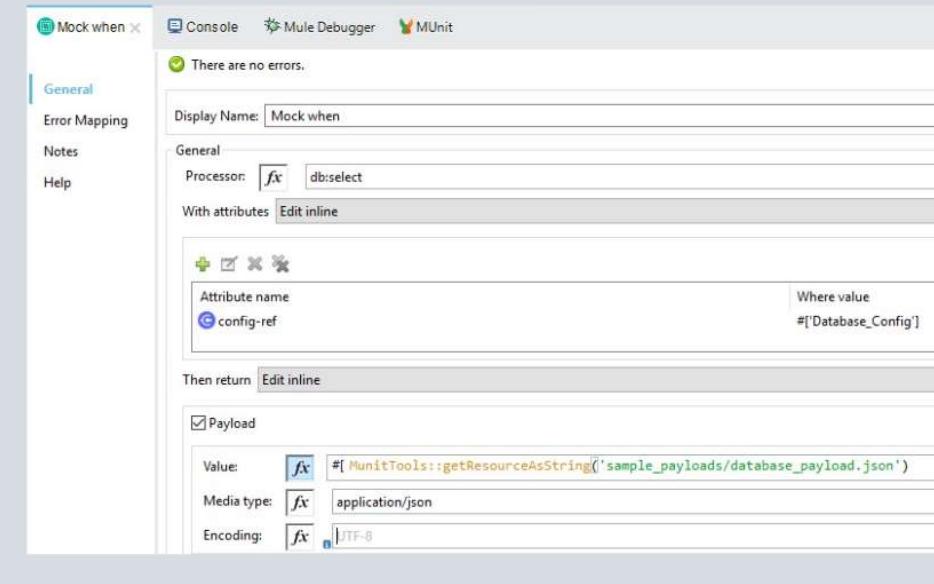
Here a value is passed as "'requestPath':'/listener'" which will point to the main flow. We can pass a value as queryParams also.



The screenshot shows the 'Set Event' configuration dialog in the MUnit interface. The 'General' tab is selected. The 'Display Name' field contains 'Set Event'. Under the 'Clone' section, there is an unchecked checkbox labeled 'Clone original event'. The 'Payload' section contains a 'Value' field with the expression '#['requestPath':'/listener']'. Below it, an 'Encoding' dropdown is set to 'JSON' and a 'Media Type' dropdown is set to 'application/json'. The 'Attributes' section is collapsed. The top bar includes tabs for 'Set Event', 'Console', 'Mule Debugger', and 'MUnit'. A status message 'There are no errors.' is displayed above the configuration fields.

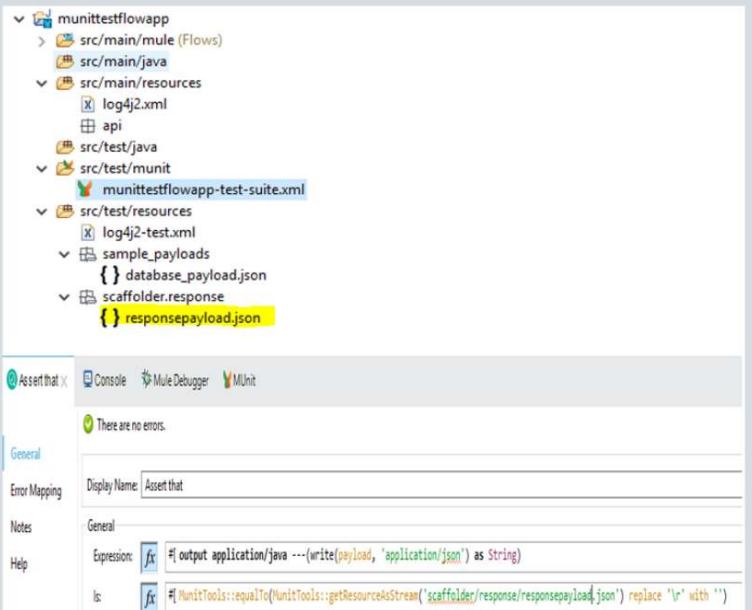
## Mock When

The 'Mock When' processor allows you to mock an Event Processor when it matches the defined name and attributes. We can use Mock when the message processor is having different scenarios. For example, let us assume we developed a Mule application and required to be tested. If the database in this application and the connector used in the message processor are not ready to accept any request, we still want to test it. In some cases, Mock When is required for a database connector.



# Assert that

- The Assert That event processor **allows you to run assertions to validate the state of a Mule event's content.**



## Assert That

The 'Assert That' event processor allows you to run assertions in order to validate the state of a Mule event's content. This is an event processor that is used to validate the Mule event after the production code runs.

For example, to assert that a payload is equal to a certain value, you can configure the Assert-That processor using the `equalTo()` matcher.

Here "Expression" works as an actual payload and "Is" works as an expected payload. So, the actual payload will compare with an expected payload.

For expected payload we have passed dummy response as:

**"MunitTools::equalTo(MunitTools::getResourceAsStream('scaffolder/response/responsepayload.json') replace '\r' with '')"** which is located inside the

## Verify Call

This processor is defined to verify if any processor was called. Here we can validate if a specific message processor has been called by using a set of attributes with a specific number of times.

- **Processor:** Describes which event processor you want to mock. The description takes the form {name-space}:{event-processor-name}. It supports regular expressions.
- **Times:** It defines the verification as successful if the event processor was called  $N$  number of times.
- **At least:** Defines the verification as successful if the event processor was called a minimum of  $N$ number of times.
- **At most:** Defines the verification as successful if the event processor was called a maximum of  $N$ number of times.

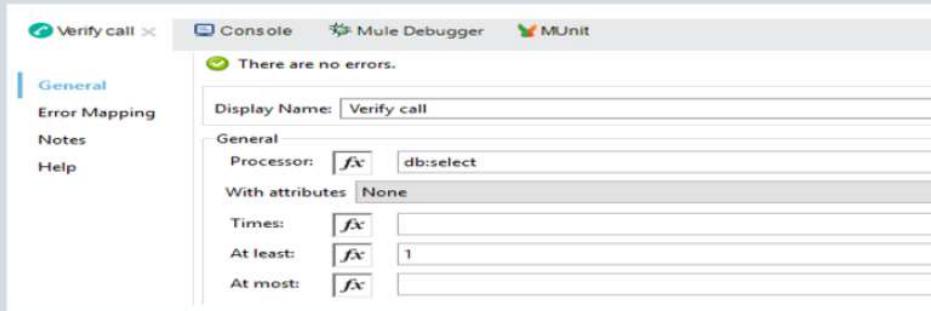
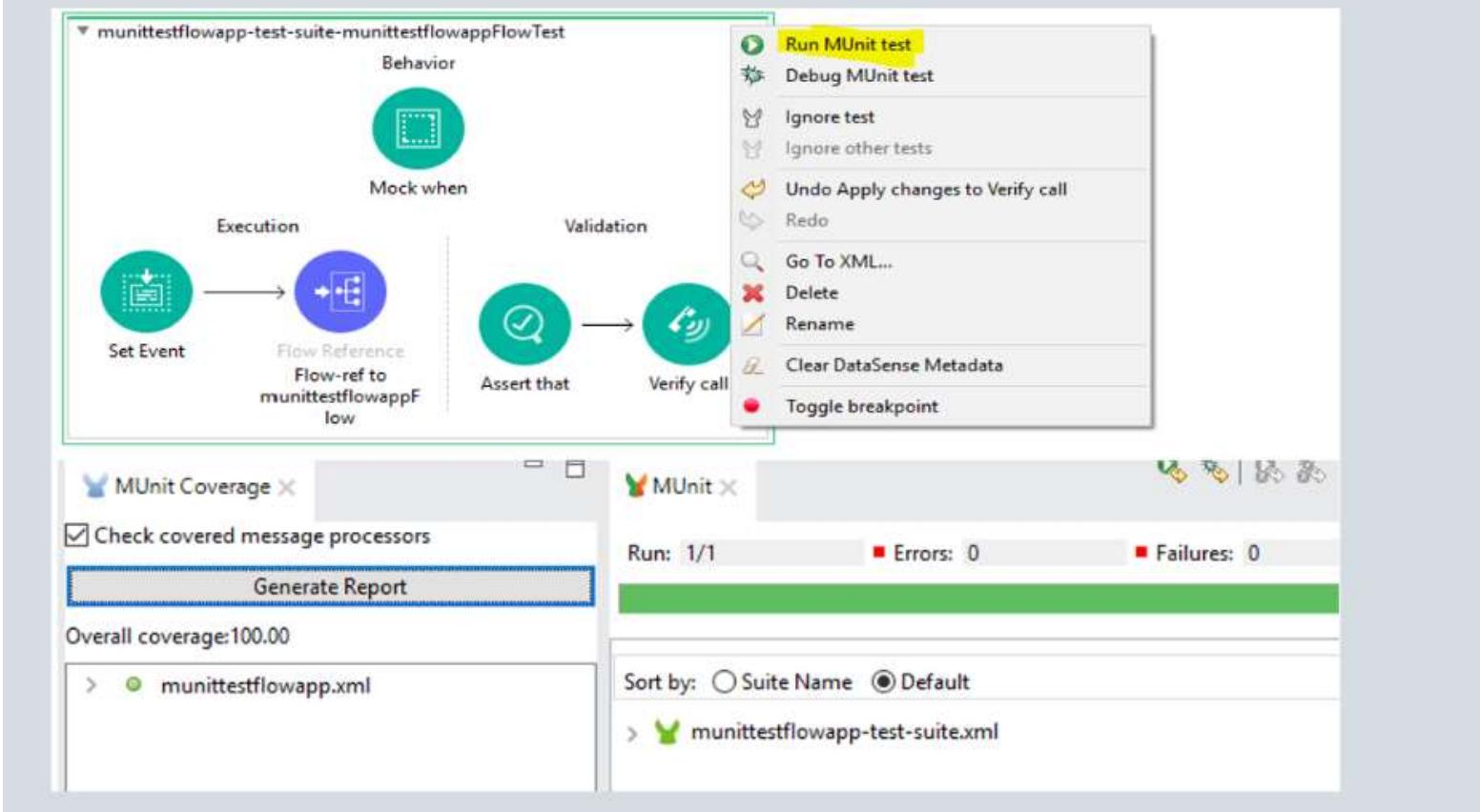


Figure 7: Verify Call structure and its use

## How to Run a Munit Test

Just right click on the Munit test and Run Munit test, as demonstrated below:



# More Important topics

HTTP Status codes	Postman	Git	Maven basics	Worker	Scaling in Mulesoft
Custom connector / Custom policies	Domain project	Cloudhub vs onpremises	Polling frequency	Load balancing	Deployment models in mulesoft
\$, \$\$, \$\$\$	Ajile and Waterfall methodology	Scrum master and Sprint	Lookup in DW	Implementation url	Message enricher & Target Variable
Object store	Rate limiting	Spike control	DB configurations	Proxy API	API Kit router

# Important Things



Register training.mulesoft.com: <https://training.mulesoft.com/home>



Install Database (My SQL ) in Local machine



Link for MuleSoft Notes: <https://www.tutorialspoint.com/mulesoft/index.htm>



Link for Mulesoft Website: <https://www.mulesoft.com/>

# Thank You

SAI KIRAN YALAKALA

