Mule 4 - Training for Absolute Beginners

➢MODULE 1: Introduction to Mule

What is MuleSoft? :

➢MuleSoft is an Integration and API Platform which helps us to :

- ➢Build and consume Web-services (RESTful & Soap)
- ➢Helps to connect different systems (Integration)
- ➢Perform Scheduler / Batch Jobs
- ➢Ability to deploy applications on- Cloudhub/ On-prem

Interview question

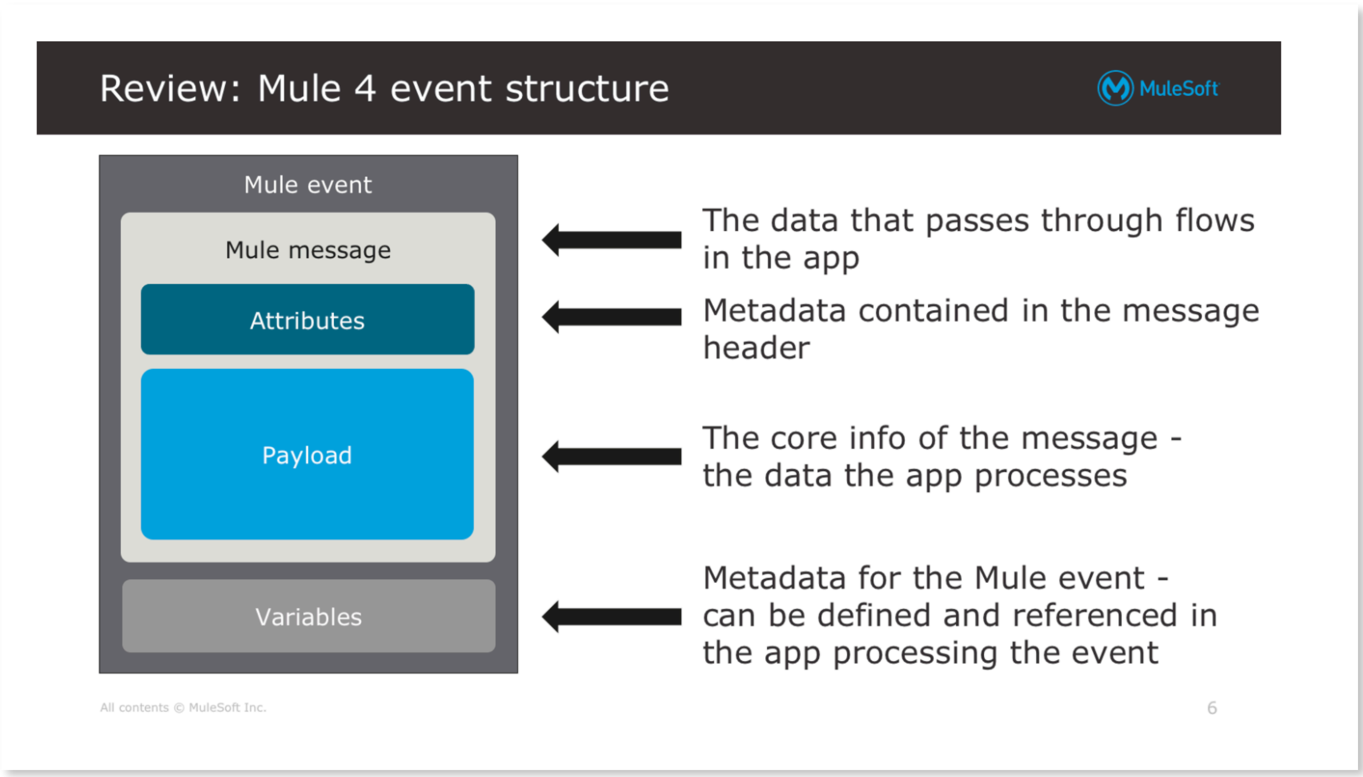
API -Led Connectivity Approach:

API Led connectivity approach is widely successful approach in building API's . The concept of System - Process - Experience API's has achieved to build reusable assets and highly secured.

In a simple way —>

- **System API's :** Are API's built to connect through underlying systems (Any system which contains the raw data) .
- **Process API's:** Process API's are built to compose or combine two or more System API's to do any kind of transformation or to implement any kind of business logics.
- **Experience API's :** These are API's which are a kind of a wrapper either to Process API's (if present) or to System API's , so that client/consumers will not have exposure or contact with direct Raw Data. (Secure purpose).

Mule Message Structure.



Review: Mule 4 event structure

- The data that passes through flows in the app
- Metadata contained in the message header
- The core info of the message - the data the app processes
- Metadata for the Mule event - can be defined and referenced in the app processing the event

Mule Event = Mule Message (payload and attributes) + Variables

Interview question

Payload : It's the actual message content. Payload can be over ridden

Attributes : These provides metadata such as queryParams, file size etc . These are immutable (cannot be changed)

Variables : These are which you can store some kind of data or information and can be used across the application as long as you connect the flows with flow-ref

Interview question

One of The Differences between Mule 4 and Mule 3:

In Mule 4 we have attributes where in Mule 3 we have inboundProperties

Eg: To access a query Param say "name"

Mule 3: message.inboundProperties.queryParams.name

Mule 4: attributes.queryParams.name

Mule 4 has only one kind of variable called "variable" where in Mule 3 we used to have 3 kinds of variables .

Eg:

Mule 3 : flowVars.name , sessionVars.role , recordVars.company

Mule 4 : vars.name , vars.role , vars.company (it will act accordingly)

Flow , Sub-flow and Private Flow:

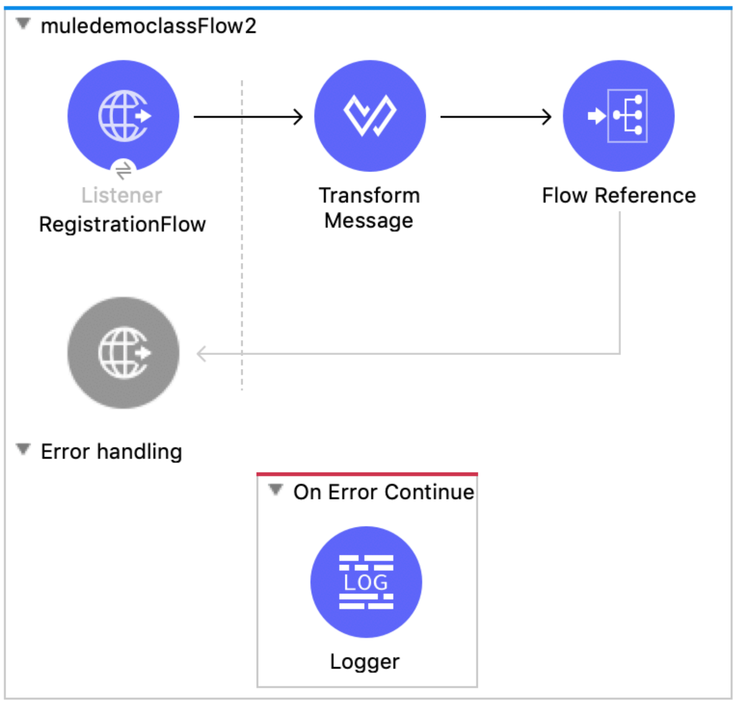
An app can consist of a single flow, or it can break up processing into discrete flows and subflows that you add to the app and connect together. Mule apps in production environments typically use multiple flows and subflows to divide the app into functional modules or for error-handling purposes.

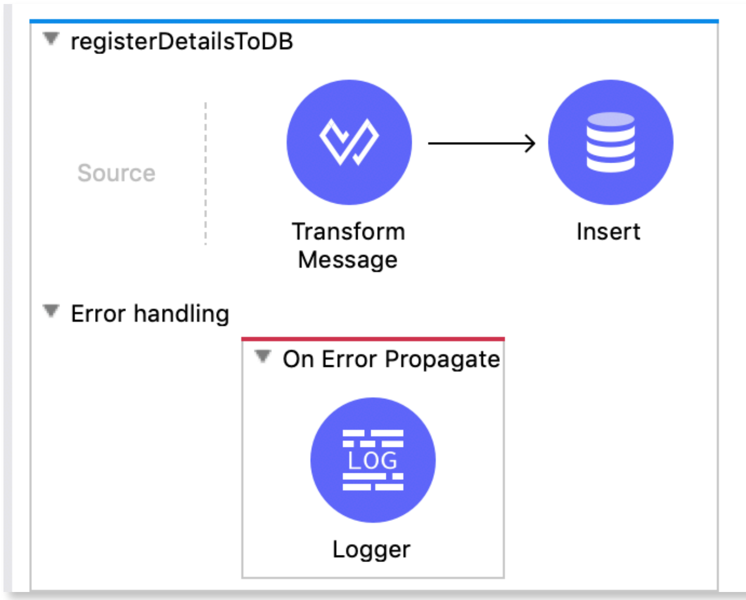
You can connect and trigger their execution with Flow Reference components or by using the DataWeave lookup function within expressions and transformations. The function passes the current event to another flow for further event processing. You can think of a Flow Reference as a function call that accepts an event as input and then returns the modified event.

Flow:

A flow has "**Source** " Part and "**Process**" Part and "**Error handling**" Part

And you will **definitely need to put something in Source**

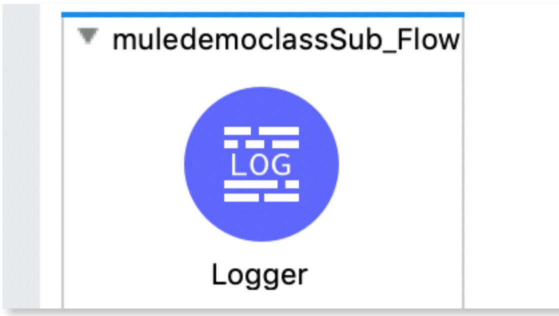




Private Flow:

A flow has "**Source** " Part and "**Process**" Part and "**Error handling**" Part

It is called a private flow when you **don't keep anything in Source .** It will be executed only if some other flow calls using flow-ref



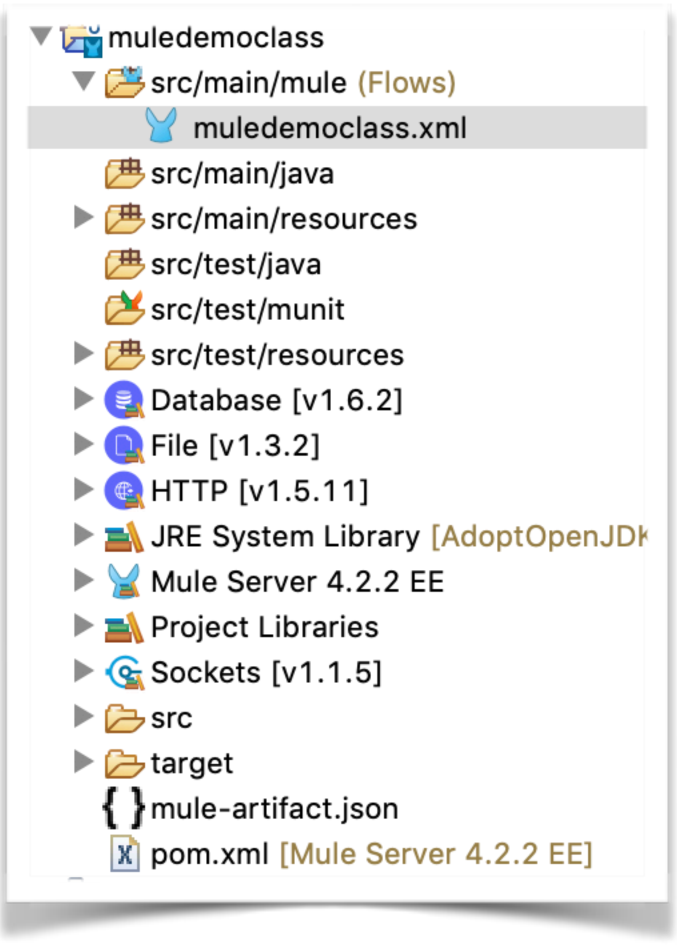
Sub-flow:

A sub flow do not have "Source" Part and don't have "Error Handling part"

It just has "Process" Part.t will be executed only if some other flow calls using flow-ref

-

Mule Project Structure:



```
▼ muledemoclass
  ▼ src/main/mule (Flows)
      muledemoclass.xml
    src/main/java
  ▶ src/main/resources
    src/test/java
    src/test/munit
  ▶ src/test/resources
  ▶ Database [v1.6.2]
  ▶ File [v1.3.2]
  ▶ HTTP [v1.5.11]
  ▶ JRE System Library [AdoptOpenJDk
  ▶ Mule Server 4.2.2 EE
  ▶ Project Libraries
  ▶ Sockets [v1.1.5]
  ▶ src
  ▶ target
    mule-artifact.json
    pom.xml [Mule Server 4.2.2 EE]
```

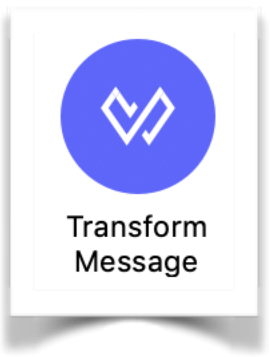
When you create a Mule Project, the folder structure looks like above

- Every project in Mule is Mavenized I.e, we use pom.xml to download all dependencies that are needed .
- All the Mule Application files are placed under **src/main/mule**
- Any other files can be placed under **src/main/resources** or **src/test/resources** based on need
- mule-artifact.json by default has “minMuleVersion” . We have **Mule Pallet** where we find connectors
- The Mule application has 3 tabs.
    - Message flow (graphical view)
    - Global Elements (contains all config details)
    - configuration xml (the xml version of graphical view)

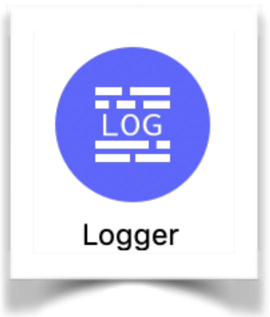
Connectors:

- **HTTP Listener :**
  - **Mandatory Config Details required:** host , port , path
  - **Default :** Accepts all kind of Methods if nothing is specified

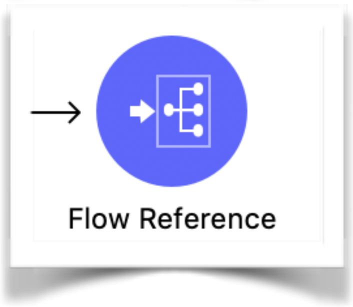
- **Set Payload :**
  - **Mandatory Config Details required:** value eg : #["Hello World"] or #[payload] or #[attributes.queryParams.name]
  - **Default :** #[payload]



- **Transform Message:**
  - **Mandatory Config Details required:** Should have something written. You can set a payload, variable or attribute using Transform message. You can set Multiple variables along with payload . Best practise is to define output type
  - **Default :** output application/java - - - { }

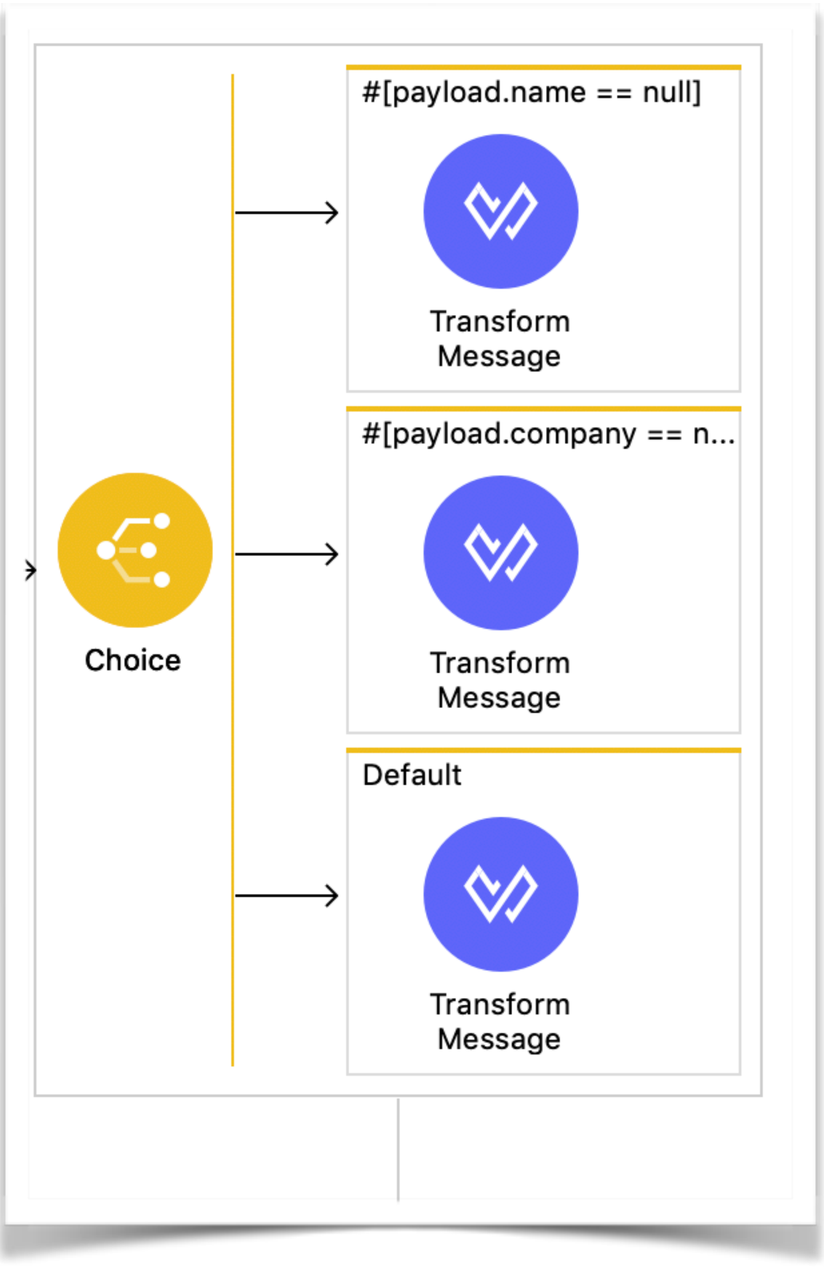


- **Logger :**
  - **Mandatory Config Details required:** Nothing is mandatory here. You can set anything in "message" field
  - **Default :** empty message



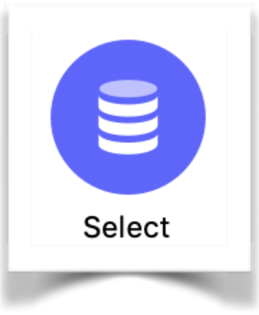
- **Flow-ref :**
  - **Mandatory Config Details required:** flow name to be called. All names are present in drop-down list
  - **Default :** empty but it fails to deploy if you don't give any flow name from drop down

- **Choice :**
    - **Mandatory Config Details required:** Expression is mandatory which checks some validation. The result of the expression should be either true or false nothing else.
    - **Expression : empty but you can't keep empty expression, it will not deploy your application**
    - **Remember :** If first condition is satisfied, then it will not go to other blocks. It will execute the first satisfied condition block and continue further



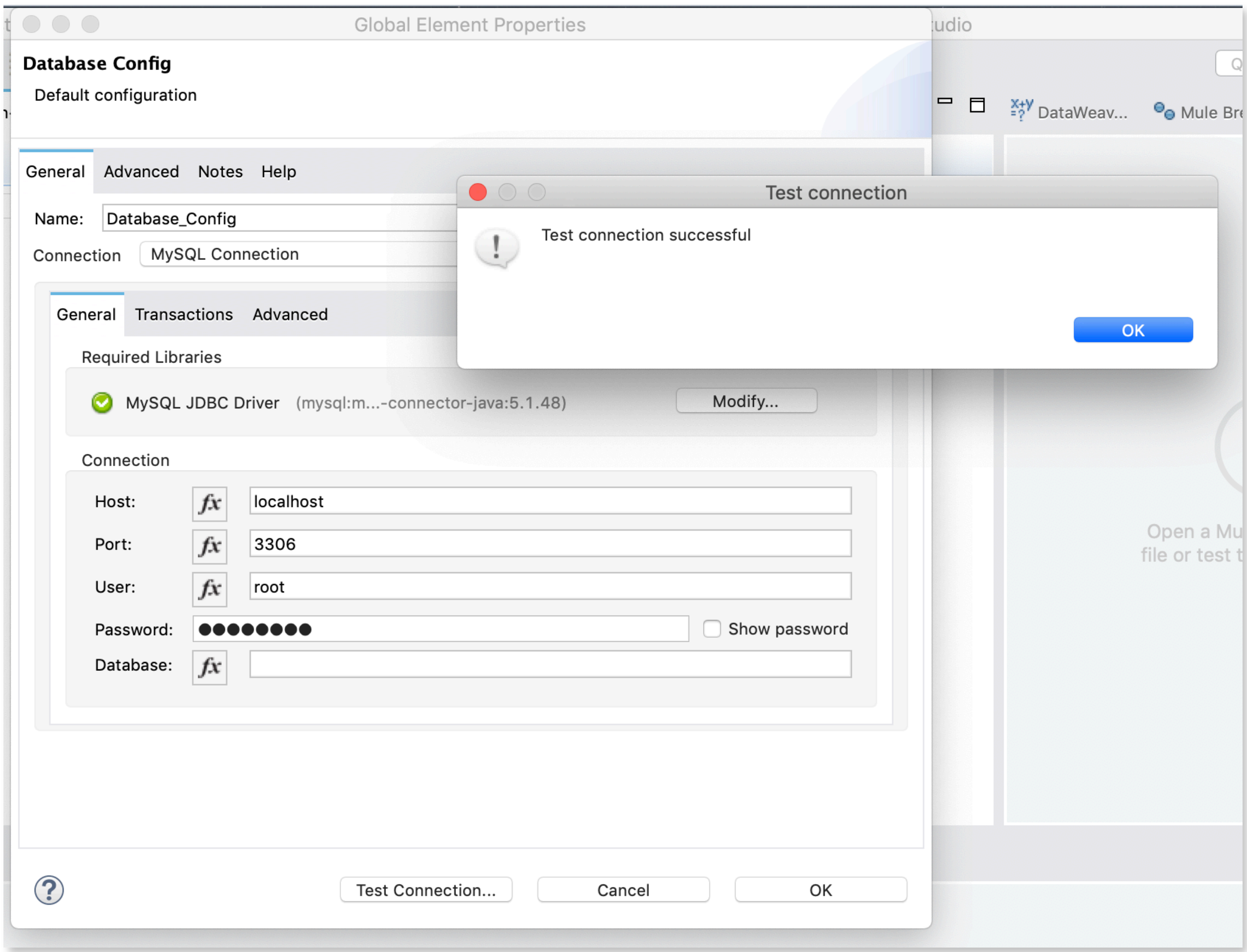
Database :

- **Mandatory Config Details required:**
  - Driver
  - Host
  - Port
  - User
  - Password
  - Database/servicename/instance
  - SQL Query text



- **Default : empty but you can't keep empty expression, it will not deploy your application**

- **Remember : In Mule 3 , we used to have one Database connector and we will be selecting the kind of operation that we want . Like Select, update, delete, insert etc. But in Mule 4 , we have separate connectors for each operation. You can test connection if you want to check if connection is successful.**



Global Element Properties

Database Config

Default configuration

General Advanced Notes Help

Name: Database_Config

Connection MySQL Connection

General Transactions Advanced

Required Libraries

MySQL JDBC Driver (mysql:m...-connector-java:5.1.48) Modify...

Connection

Host: localhost

Port: 3306

User: root

Password: ●●●●●●●● Show password

Database:

Test Connection... Cancel OK

Test connection

Test connection successful

OK

The **Output of Database is Array (applications/java).**

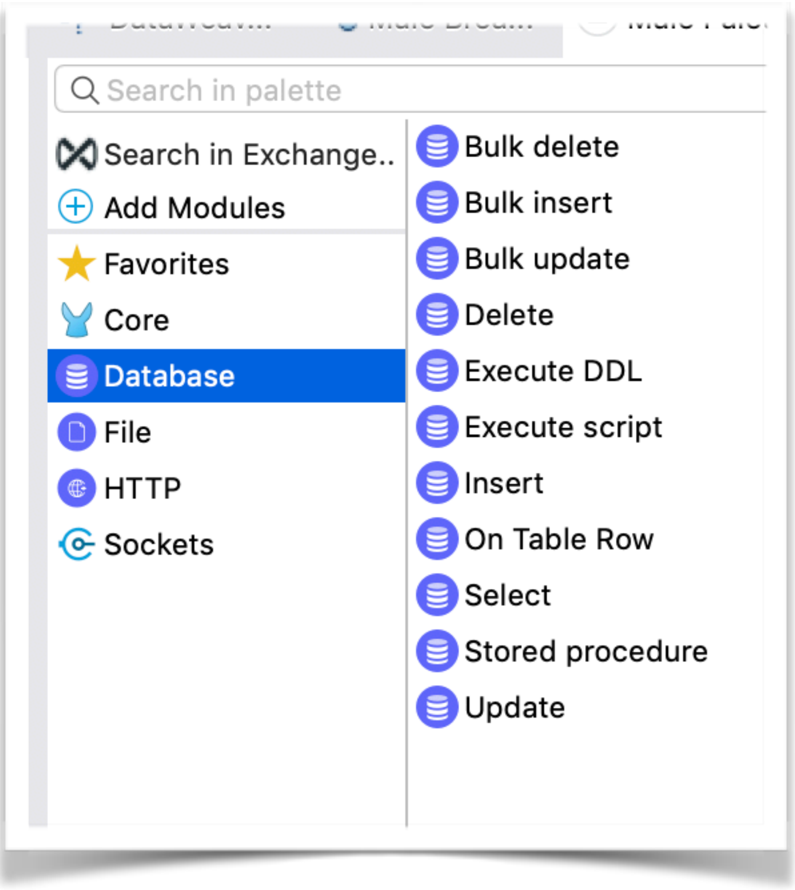
Convert to other Data Types like json , csv, xml or any to Display or return response to caller. Because you cannot print an Object. If you want to use internal purpose there is no need to do conversion.

Sample Output from Select Database connector :

Array of Objects:

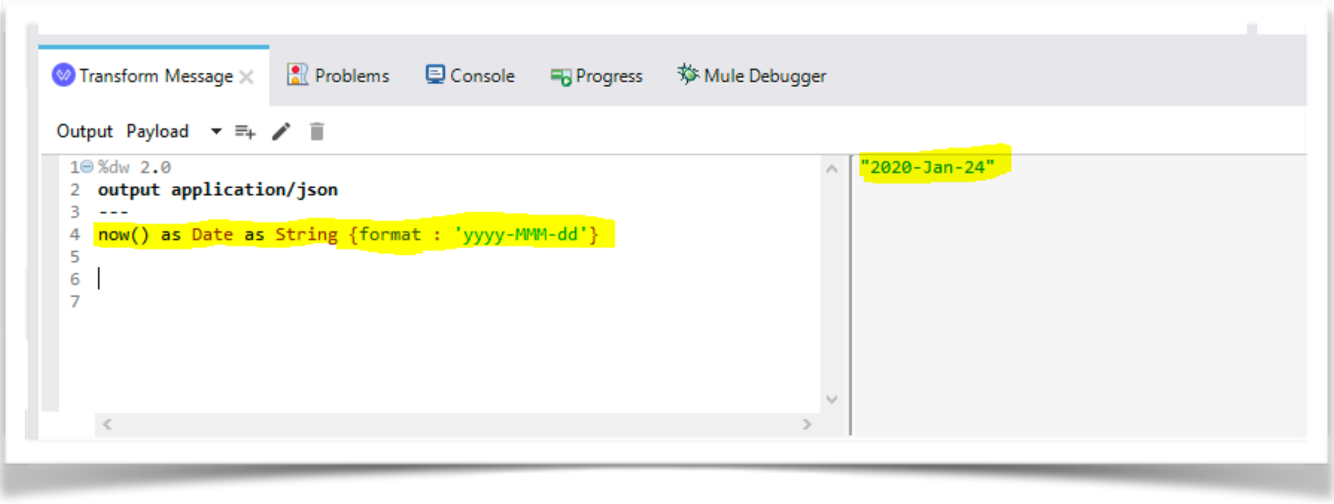
```
[
        {
                "name" : "Sravan",
                "id"     :  2
        },
        {
                "name" : "Lingam",
                "id"     :  3
        }
]
```

Operations :



The Power of Transform Message - Preview Mode :

Refer to : https://www.linkedin.com/pulse/dataweave-20-part-2-power-transform-message-preview-sravan-lingam/



```
%dw 2.0
output application/json
---
now() as Date as String {format : 'yyyy-MMM-dd'}
```

```
"2020-Jan-24"
```

Do you know?

How to Identify an Object and an Array?

Simple :

An Object is always enclosed with { } and contains key-value pair

Eg: { “name” : “Sravan” } — Valid √

{ “name” : “Sravan” , “id” : “2”} — Valid √

{ “Sravan” } — InValid X **{ “name” : “Sravan” , “2”} — InValid X**

An Array is always enclosed with [ ] and contains only values Eg:

[“Sravan” , “2” ,”MuleSoft”] — Valid √

{ “name” : [“Sravan”, “Lingam”]} — Valid √

[“name” : “Sravan”] — InValid X

Building Simple RESTful Application to print “Hello World” :

Step 1 : Create a new project . File —> New —> Mule Project

Step 2 : Drag and Drop a Flow from Mule Pallet

Step 3 : In “Source” part of Flow , drag and drop HTTP Listener and enter basic config details (Refer to Connectors section) like host, port, and path. (remember base path and normal path are different)

Step 4 : Drag and Drop Set-payload or Transform Message component , both hold same value . And type “Hello World” in value.

Step 5: Place a logger component to print your message in console

Step 6 : Right click on Canvas and click Deploy application

Step 7 : Goto Postman and create url and deploy . URL can be frame like below:

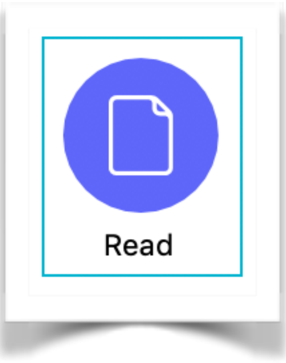
<Protocol> :// <host>:<port> /<basePath>/<path>

In our case : Protocol : http , port : 8081 , basePath : “blank” , path : test

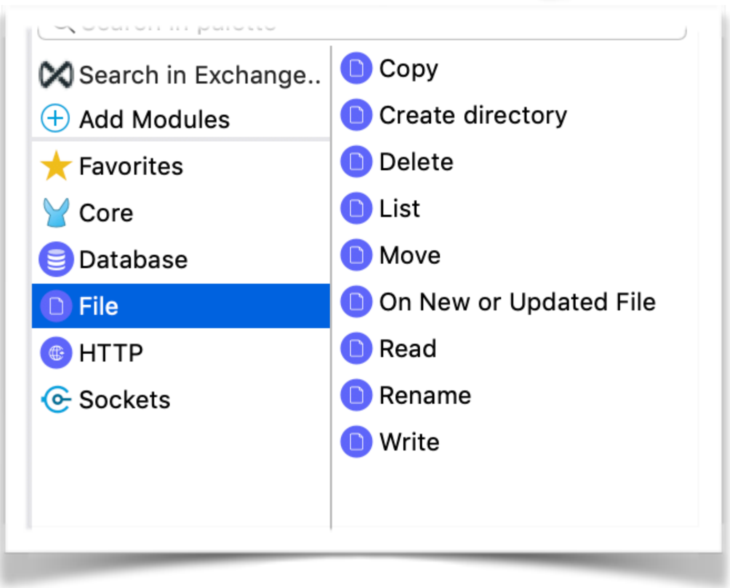
URL : **http://localhost:8081/test**

- **File :**

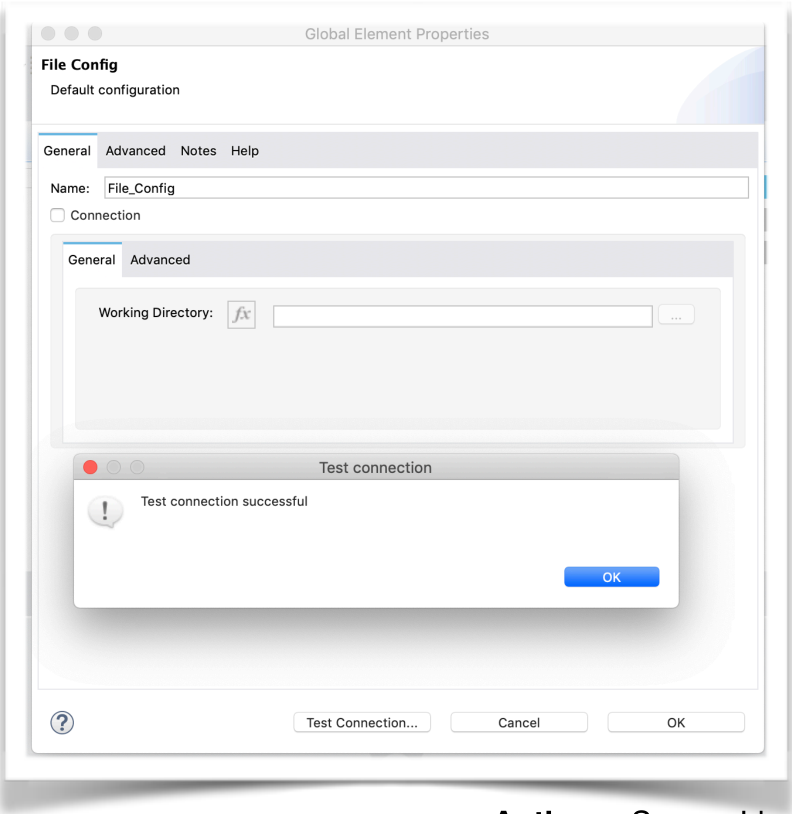
  - **Mandatory Config Details required:** Only configuration which has one value that needs to be set I.e, Working Directory. You can leave empty too. But You have to create a Configuration for sure.



  - **Operations:**



Test Connection:


Global Element Properties

File Config

Default configuration

General Advanced Notes Help

Name: File_Config

Connection

General Advanced

Working Directory:

Test connection

Test connection successful

OK

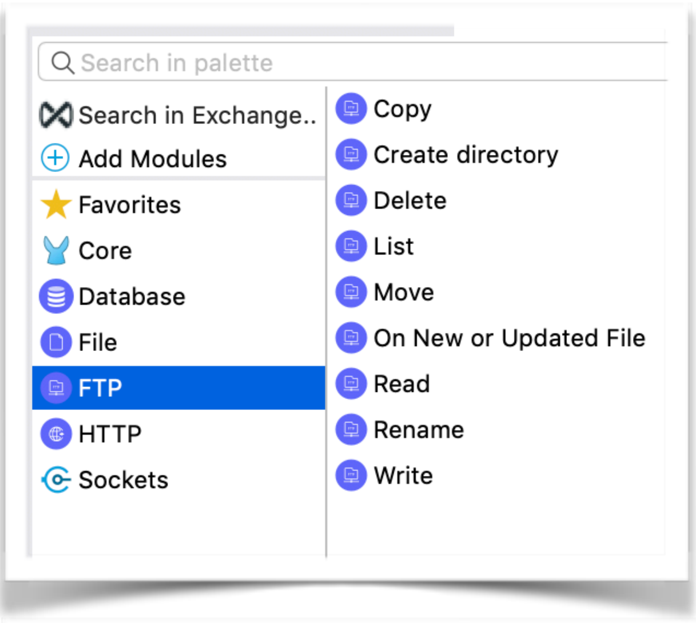
Test Connection... Cancel OK

- **FTP / SFTP :**

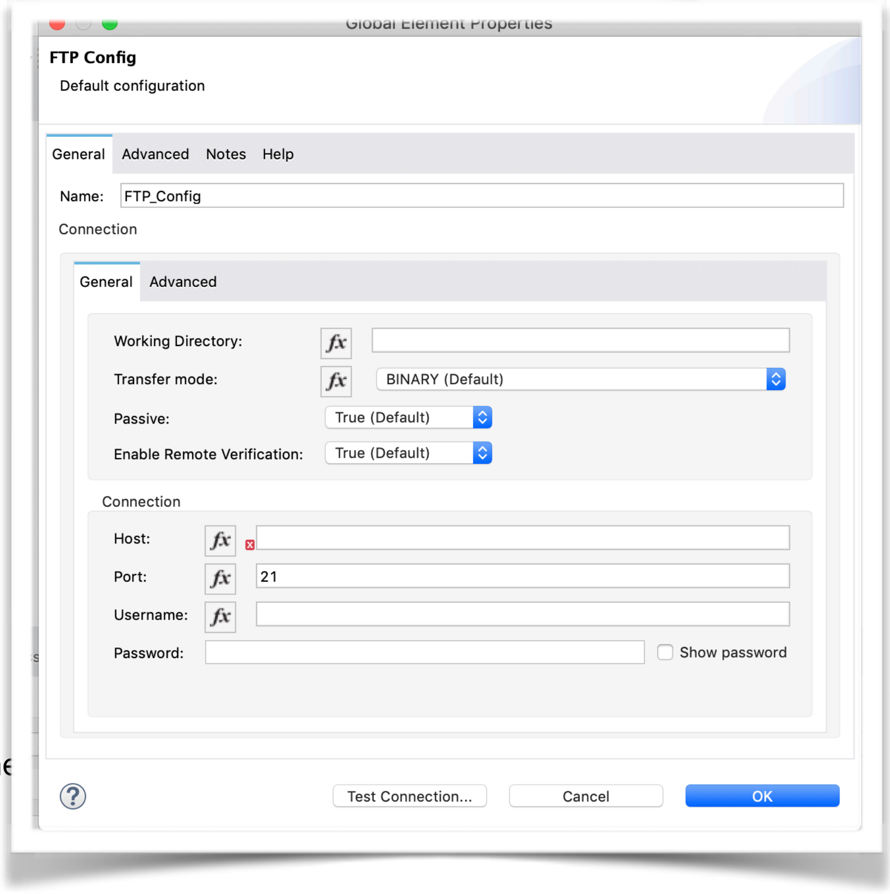
  - **FTP, SFTP are one and same except that SFTP uses secured protocol. The way we connect, the operations are all same. The operations and the way it works is same for File,FTP and SFTP .**

  - **Mandatory Config Details required:** Host, port , username and password

  - **Operations:**



FTP Config:


Global Element Properties

FTP Config
Default configuration

General | Advanced | Notes | Help

Name: FTP_Config

Connection

General | Advanced

Working Directory:
Transfer mode: BINARY (Default)
Passive: True (Default)
Enable Remote Verification: True (Default)

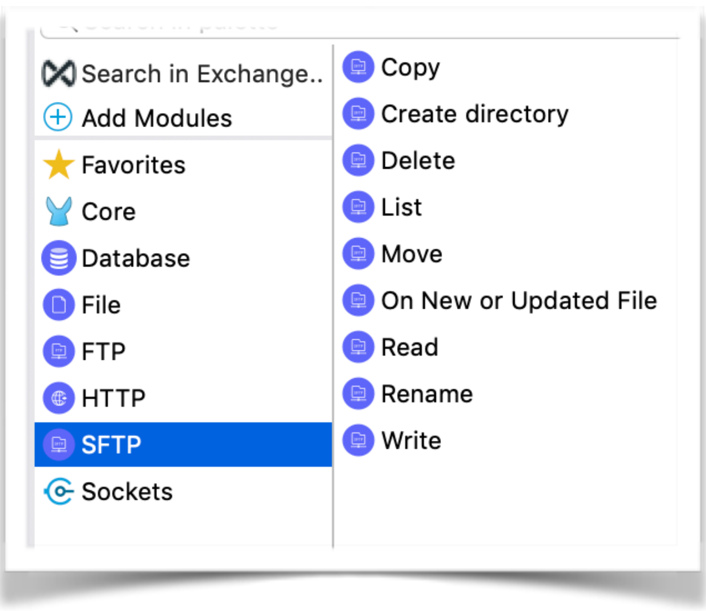
Connection

Host:
Port: 21
Username:
Password:
Show password

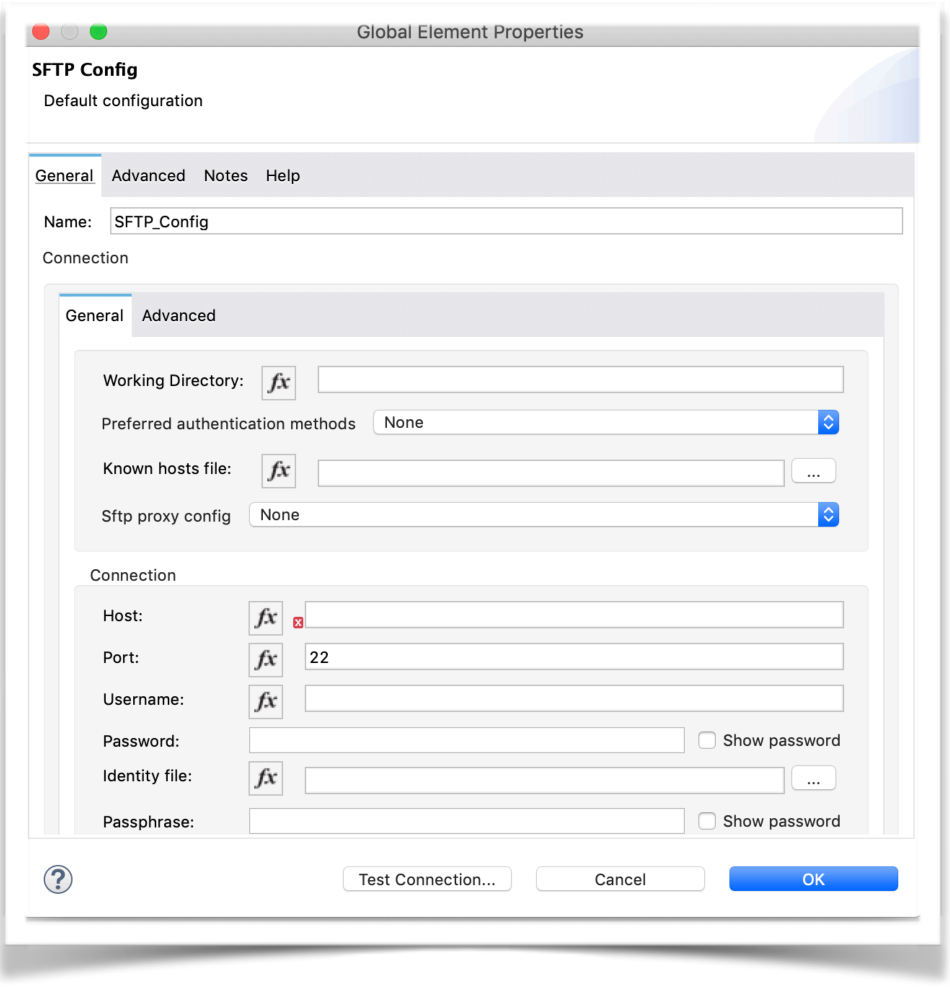
Test Connection... | Cancel | OK

SFTP:

- **Operations:**



SFTP Config:


Global Element Properties

SFTP Config
Default configuration

General | Advanced | Notes | Help

Name: SFTP_Config

Connection

General | Advanced

Working Directory:
Preferred authentication methods: None
Known hosts file:
Sftp proxy config: None

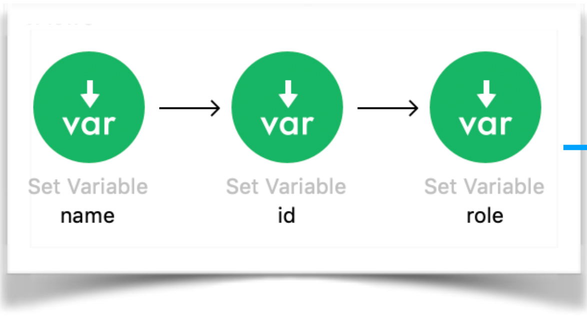
Connection

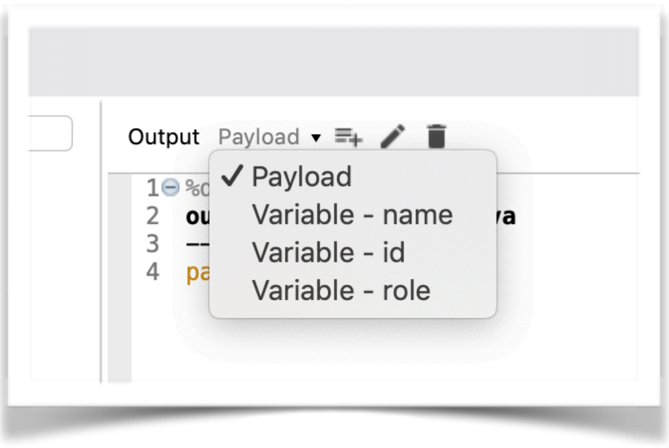
Host:
Port: 22
Username:
Password: ☐ Show password
Identity file:
Passphrase: ☐ Show password

If you have observed above : File,FTP and SFTP operations, all are same because the functionality remains same. Only difference is File is on local, FTP and SFTP are on Server.

Do you know?

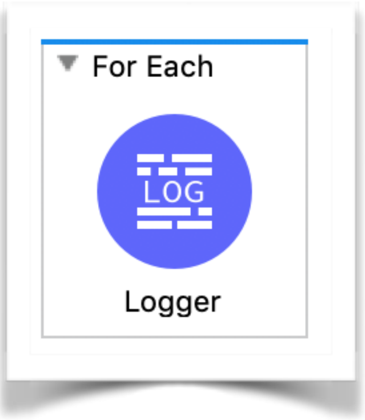
We can set **multiple variables , Multiple Attributes including payload** in Single Transform Message. So that you can avoid having multiple "Set variable" components in a flow.

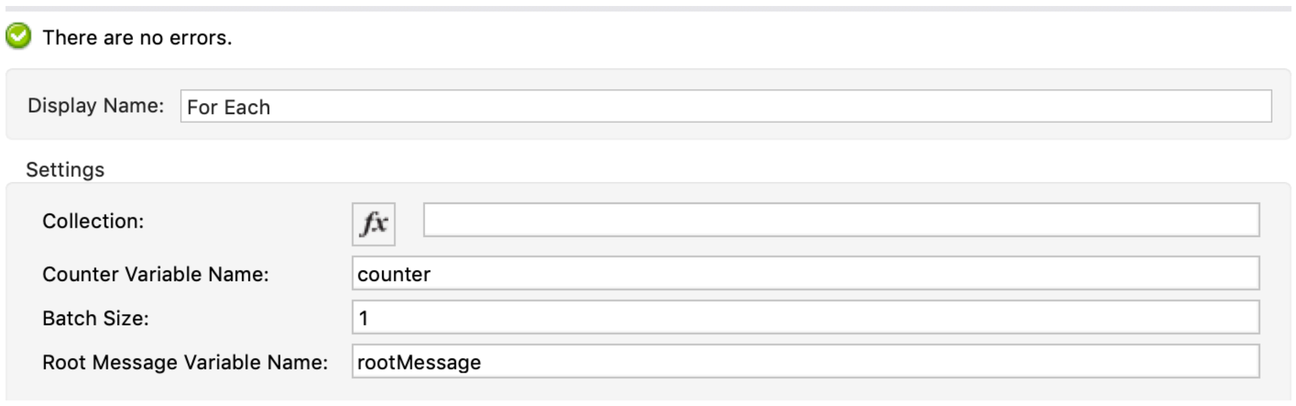




For Each :

- **Mandatory Config Details required:** nothing
- **Default :** By default if you don't mention anything in "Collection" Tab, it takes payload as collection. Batch size=1
- **Remember**: Input collection for For-each must be always an Arrays





There are no errors.

Display Name:	For Each

Settings

Collection:	
Counter Variable Name:	counter
Batch Size:	1
Root Message Variable Name:	rootMessage

Interview questions

Note:

1) The payload value After For-each Loop is always the payload value that is present just before For each.

2) Everything is a payload of single object once it enters For-each loop . Even though if your "collection " is vars or payload. As For-each iterates over every single record of "collection" . The payload value is always single object inside for-each loop

3) Input for For-each is payload by default if nothing is mentioned in "Collection" and it should be array.

4) You cannot keep empty For-each loop. It will fail to deploy if there's empty for-each

5) If one record is failed in the process, then For-each will stop processing the rest of the records. For eg: You have an array of 5 . And 1st two records are processed and somehow 3rd record has failed to due to various reasons . Then your 4th and 5th records will not be processed

6) If you want to process all records even if there are failures in individual records. Then you must use "Try" block and use"On-Error-Continue" to handle individual errors and process to next record.

7) You can keep payload or any variable value inside "Collection" tab . But it should definitely be an array

HTTP STATUS CODES :

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

- Informational responses (100–199),
- Successful responses (200–299),
- Redirects (300–399),
- Client errors (400–499),
- and Server errors (500–599).

Every Request that is made by client/consumer but receive some response along with a Status Code.

So whenever you build a. Web-service (either RESTful or SOAP) please make sure that you set appropriate Status code.

Frequently Used Status codes:

- 200 OK
- 201 Created
- 202 Accepted
- 400 Bad Request
- 401 Unauthorized
- 402 Payment Required
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 406 Not Acceptable
- 409 Conflict
- 415 Unsupported Media Type
- 500 Internal Server Error
- 501 Not Implemented
- 502 Bad Gateway
- 503 Service Unavailable
- 504 Gateway Timeout

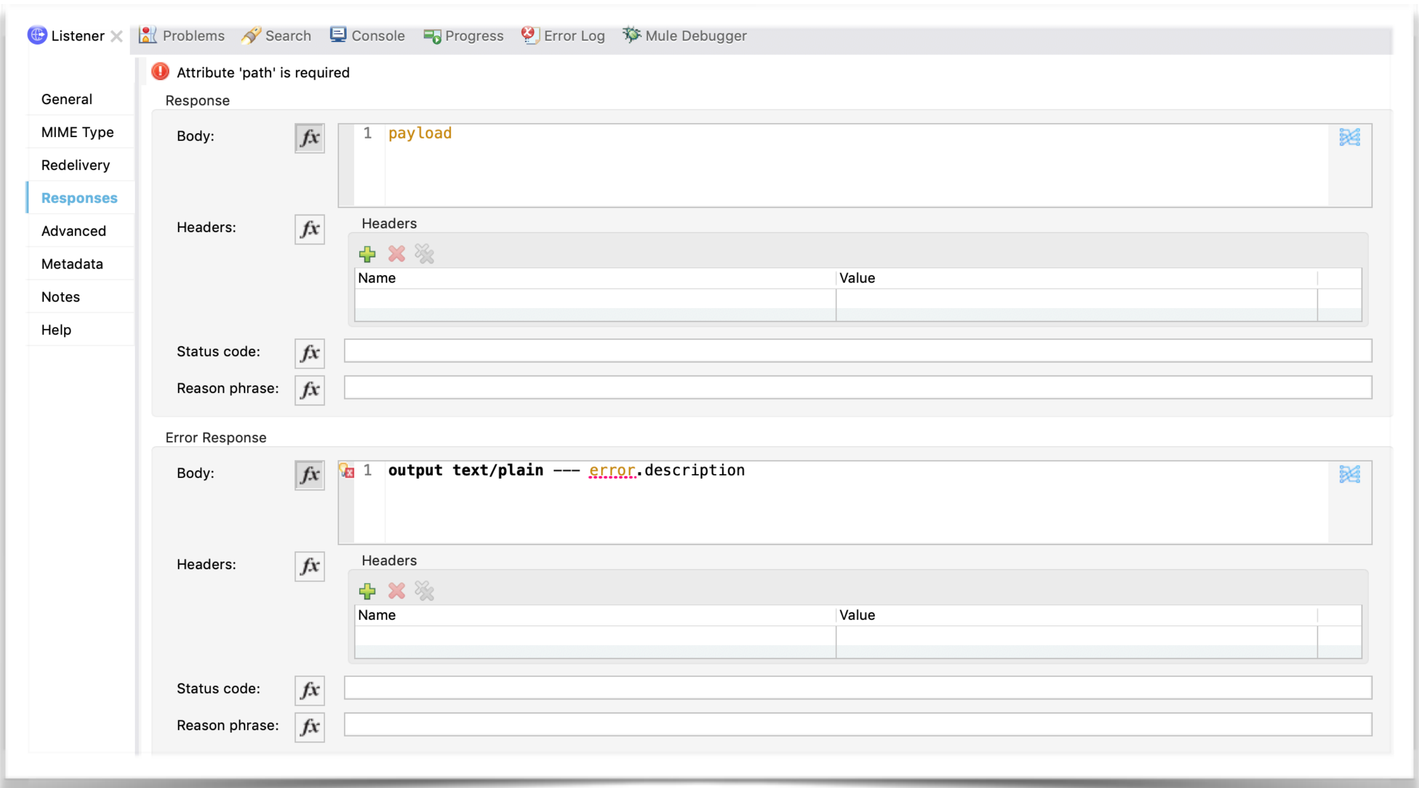
How To Set HTTP Status Code in Mule Applications ?

HTTP Listener place a very important role in setting output response on both the Body and Status code.

We shall check how the Responses Tab of Http Listener looks like by default.

Please Note: Below HTTP Listener is which manually dragged and drop by developer.

The Http Listener generated using RAML will be having different values in Responses Tab



Attribute 'path' is required

Response

Body:
```
payload
```
Headers:

Name	Value

Status code:

Reason phrase:

Error Response

Body:
```
output text/plain --- error.description
```
Headers:

Name	Value

Status code:

Reason phrase:

You can see there are 2 divisions:

1) **Response** - Which has Body and Status code. (Body is set to payload by default and can be changed if required. By default Status code is empty, i.e, default status code is 200) . Used for success scenarios

2) **Error Response** - Which has Body and Status code. (Body is set to error.description by default and can be changed if required. By default Status code is empty, i.e, default status code is 500) . Used for failure scenarios

If you want to set any status code manually. You can define any variable name in place of Status Code (Both in Response and Error Response tabs) and **make sure that Status code value is set with Same Variable name**

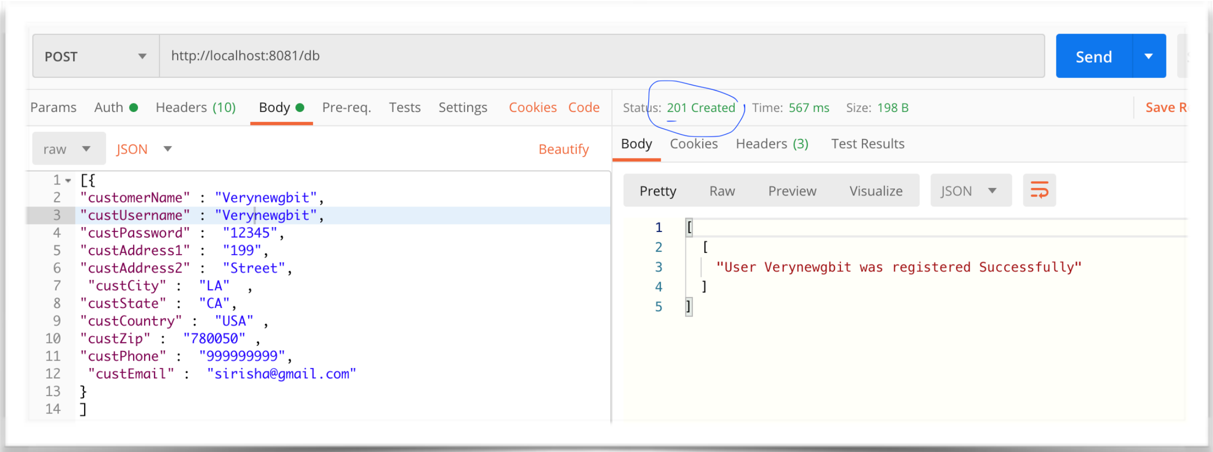
Eg: If you give Status code value as **vars.httpStatus** . Then you have to set this variable at the end of each process whenever you require to change other than 200 (for success) or 500(for failure)

Note: Mostly All RESTful Applications use RAML to generate basic flows. So no need to manually give names in Http Listener. Because by default the Status codes are named as vars.httpStatus.

If you forget to set status code value somewhere , then it will show 200 for success and 500 if any failure is present.

<u>Examples:</u>

201:

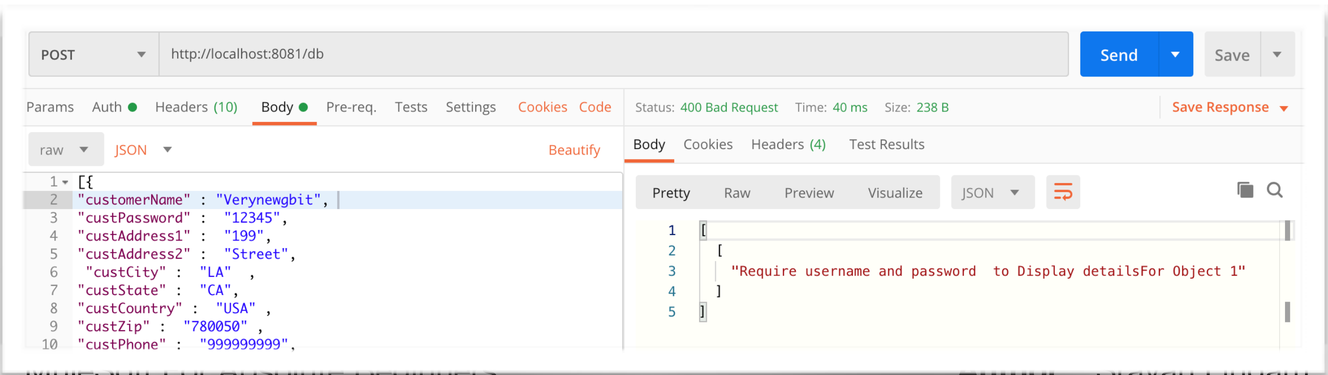

POST http://localhost:8081/db

Status: 201 Created Time: 567 ms Size: 198 B

```
[{
"customerName" : "Verynewgbit",
"custUsername" : "Verynewgbit",
"custPassword" :  "12345",
"custAddress1" :  "199",
"custAddress2" :  "Street",
 "custCity" :  "LA"  ,
"custState" :  "CA",
"custCountry" :  "USA" ,
"custZip" :  "780050" ,
"custPhone" :  "999999999",
 "custEmail" :  "sirisha@gmail.com"
}
]
```

```
[
  [
    "User Verynewgbit was registered Successfully"
  ]
]
```

400:

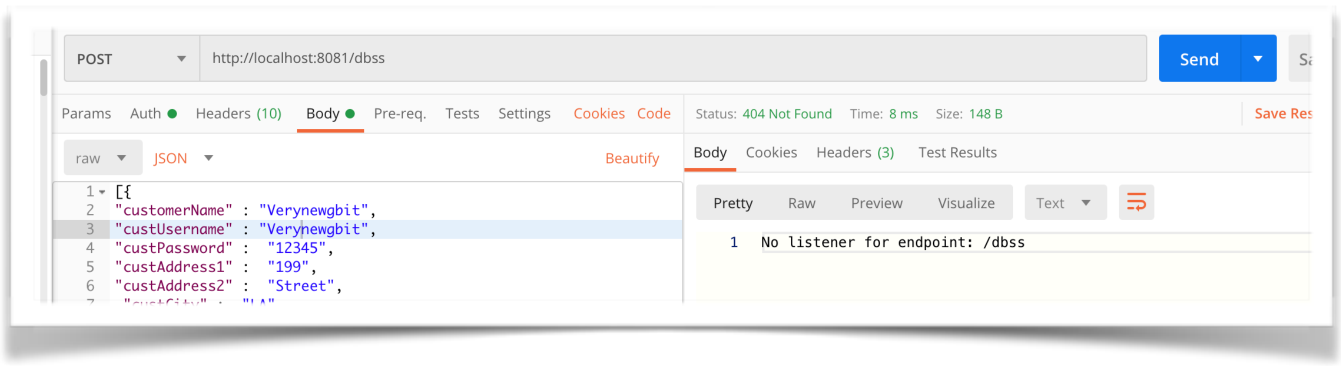

POST http://localhost:8081/db

Status: 400 Bad Request Time: 40 ms Size: 238 B

```
[{
"customerName" : "Verynewgbit",
"custPassword" :  "12345",
"custAddress1" :  "199",
"custAddress2" :  "Street",
 "custCity" :  "LA"  ,
"custState" :  "CA",
"custCountry" :  "USA" ,
"custZip" :  "780050" ,
"custPhone" :  "999999999",
```

```
[
  [
    "Require username and password  to Display detailsFor Object 1"
  ]
]
```

404:



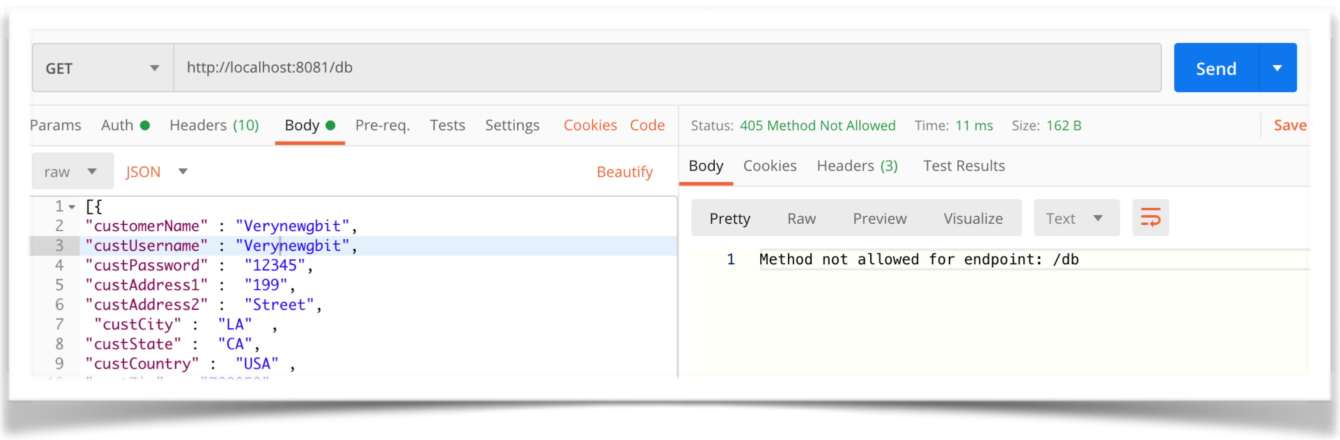
POST http://localhost:8081/dbss

Status: 404 Not Found Time: 8 ms Size: 148 B

```
[{
"customerName" : "Verynewgbit",
"custUsername" : "Verynewgbit",
"custPassword" :  "12345",
"custAddress1" :  "199",
"custAddress2" :  "Street",
```

```
No listener for endpoint: /dbss
```

405:



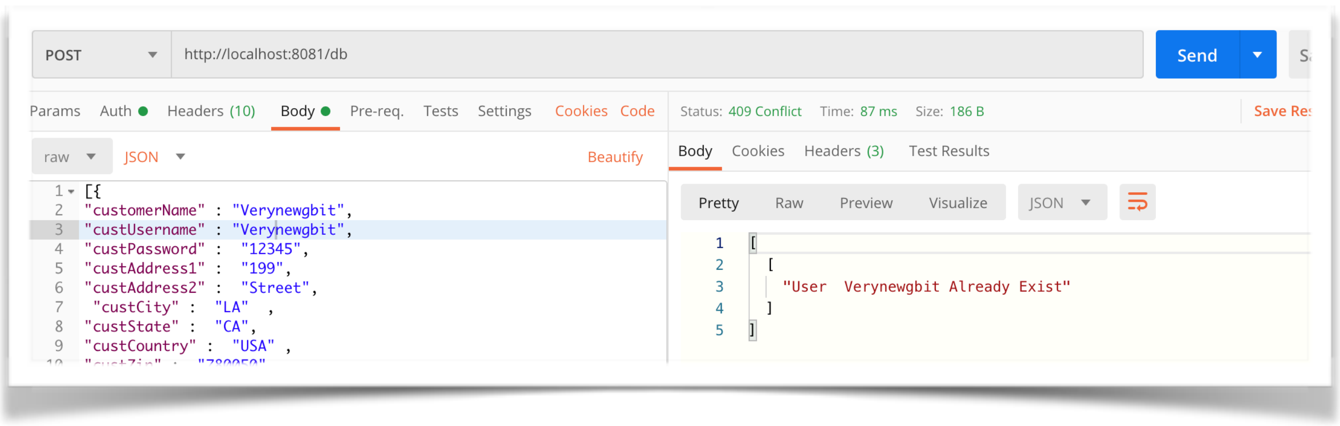
GET http://localhost:8081/db

Status: 405 Method Not Allowed Time: 11 ms Size: 162 B

```
[{
"customerName" : "Verynewgbit",
"custUsername" : "Verynewgbit",
"custPassword" :  "12345",
"custAddress1" :  "199",
"custAddress2" :  "Street",
 "custCity" :  "LA"  ,
"custState" :  "CA",
"custCountry" :  "USA" ,
```

```
Method not allowed for endpoint: /db
```

409:



POST http://localhost:8081/db

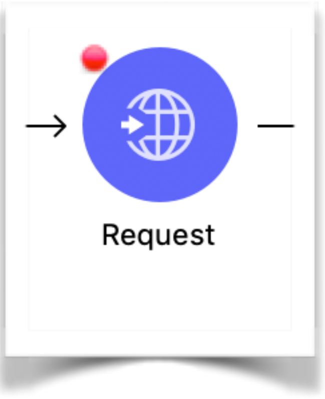
Status: 409 Conflict Time: 87 ms Size: 186 B

```
[{
"customerName" : "Verynewgbit",
"custUsername" : "Verynewgbit",
"custPassword" :  "12345",
"custAddress1" :  "199",
"custAddress2" :  "Street",
 "custCity" :  "LA"  ,
"custState" :  "CA",
"custCountry" :  "USA" ,
```

```
[
  [
    "User  Verynewgbit Already Exist"
  ]
]
```

This we we follow the standards of HTTP Status codes

Http Request :

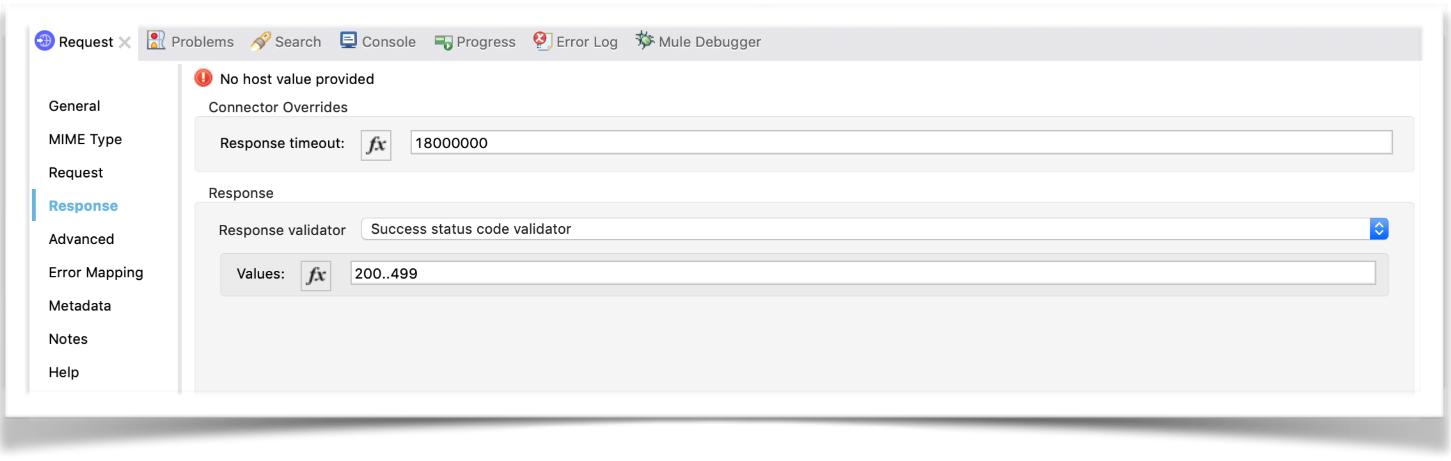


- Http Request is used to call other web-services . It can be placed only in "Process" area of flow/private-flow/sub-flow . We should make sure that we send all kind of proper inputs to Http Request component like URL,method, the input payload that the external web-service is expecting, various attributes like queryParams,headers etc

- **Mandatory Config Details required:**
  - Protocol
  - Method
  - URL or Path : If you provide whole URL, then port base path are not required. But If you give path, then host, port , path are necessary.

You can pass body , headers ,Attributes , QueryParams in respective tabs

1. In any case if an error coming from the web-service we are calling to be considered as success , then we have to make changes in Response Tab of Http Request and make use of **Success-status-code-validator**.



No host value provided

Connector Overrides

Response timeout: 18000000

Response

Response validator: Success status code validator

Values: 200..499

Example: If you are receiving 404 NOT FOUND error or any 400 series errors needs to be considered as success scenario instead of error .

In above case **200..499** states that all the response status codes pithing range 200 to 499 are need to be considered as Success

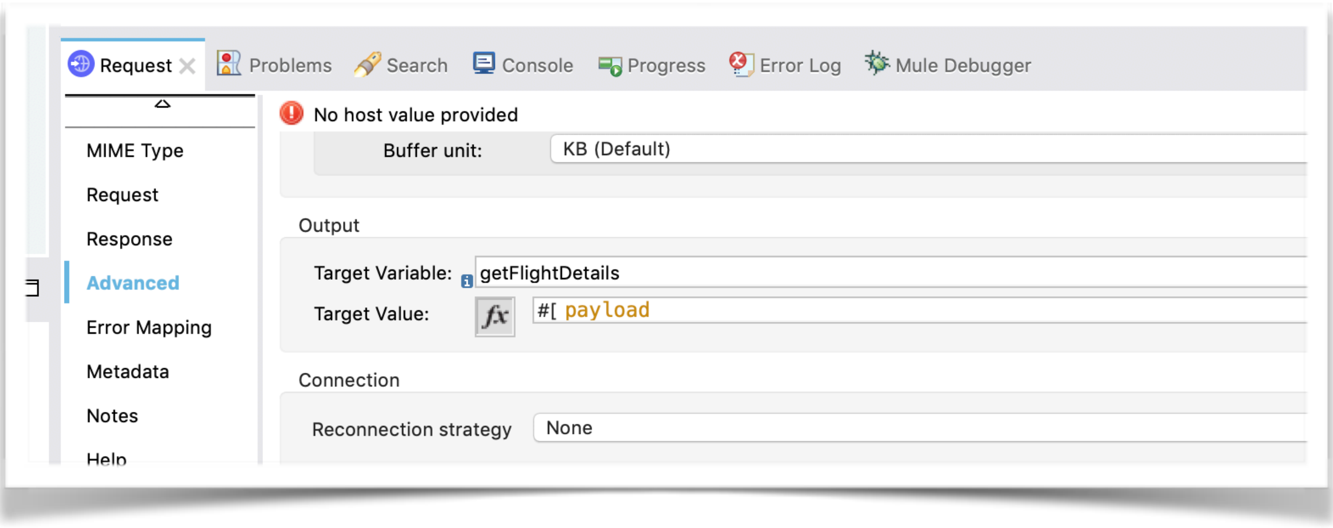
Same way, if you want to consider only one particular Status code as a failure and rest all Status codes as Success , then we make us of **Failure-Status-Code-Validator**

2) If you feel that the time taken to get response form external web-service is taking long than expected, then we can increase **Response Timeout** Value . Which is just about Response Validator configuration in Response Tab (Refer to above pic).

3) All Attributes and Payload just before Http Request are overridden by attributes and Payload that we received from external web-service via Http Request .

4) To check the status code coming form Http Request , we use #[attributes.statusCode]

5) If you still don't want to Override existing payload just before HTTP Request , we can give **Target Variable** Name which is in Advance Tab of Http Request , so that the output payload of web-service is stored in a variable that you have defined . From below picture , we can see that your payload is stored in variable named getFlightDetails and can be accessed across the applications using vars.getFlightDetails.



No host value provided

Buffer unit: KB (Default)

Output

Target Variable: getFlightDetails

Target Value: #[payload

Connection

Reconnection strategy None

● Error Handling :

An exception occurs when an unexpected event happens while processing.
Exception handling is the process of responding to exceptions when a computer program runs.

In Mule we can handle the message exception at different level.

- –At Project level using **Default error handler**
- –At Project level using **Custom Global error handler**
- –At Flow level in exception handling using **On Error Continue** and **On Error Propagate, Raise Error**
- –With in Flow or at processor level using **try scope**

Whenever there is an error occurred in a Flow, an **error Object** is created . It contains many properties . Eg : **error.description** , **error.errorType** etc

errorType is combination of Namespace and Identifier :
Eg : **HTTP**:**UNAUTHORIZED**

Here **:**
namespace = **HTTP**
Identifier = **UNAUTHORIZED**

Mule identifies based on error Type and then route to respective blocks that are placed in Error Handler.

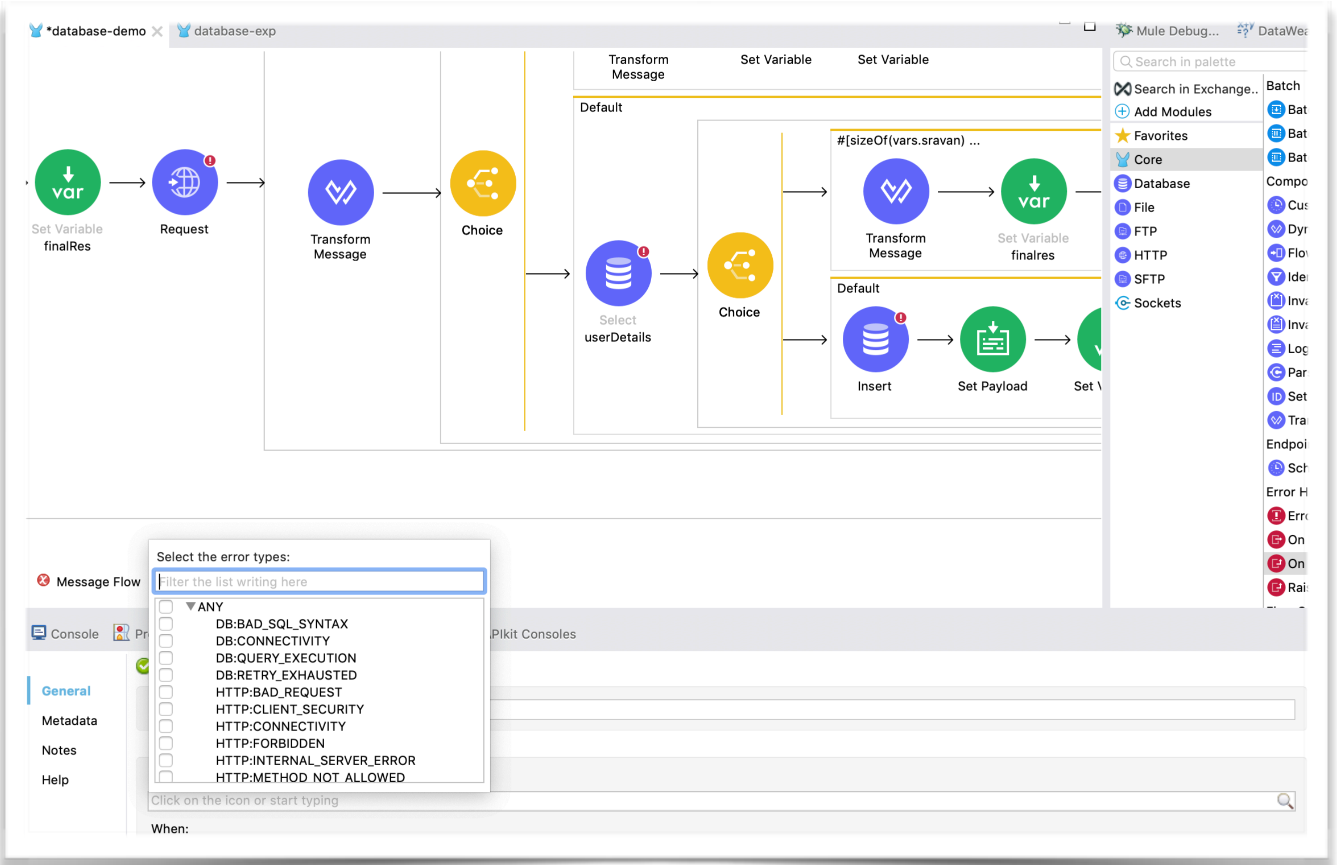
Let’s discuss what is On Error Propagate and On Error Continue in detail:

As we have discussed earlier about the differences between flow , sub-flow and private flow, Sub-Flow doesn’t have Error Handling scope . So its just Flow and a Private flow where you can place On-Error-Propagate and On-Error-Continue inside Error-handling block.
As we discuss earlier , either it is Propagate or Continue, Mule Executes all the components within that block

We use to have many kinds of error-handling connectors in Mule 3 . But it was simplified in Mule 4

Remember : The error will route to error-handling only if it identifies that the errorType of that error is handled

Also Mule 4 has one excellent feature of Identifying the types of errors that can occur within that flow by looking at what kind of connectors are placed in that particular flow.



Select the error types:

Filter the list writing here

```
▼ANY
   DB:BAD_SQL_SYNTAX
   DB:CONNECTIVITY
   DB:QUERY_EXECUTION
   DB:RETRY_EXHAUSTED
   HTTP:BAD_REQUEST
   HTTP:CLIENT_SECURITY
   HTTP:CONNECTIVITY
   HTTP:FORBIDDEN
   HTTP:INTERNAL_SERVER_ERROR
   HTTP:METHOD_NOT_ALLOWED
```

Click on the icon or start typing

When:

You can see above that as we have Http-Request and Database , it has all kind of error types which are expected to come in error scenarios . Even you can find EXPRESSION error type as we have some Dataweave syntaxes in the flow.

It will be very easy to learn Error-handling on how Error-Propagate or Error-Continue works . We shall follow some set of rules to identify the flow of process. That way we can determine what is the result payload statusCode

Before going to learn about rules. We must not forget that , RAML based generated Http Listener has Error Response Body as "payload" as Default but manually dragged and drop Http Listeners has **output text/plain - - - error.description** as default. So change accordingly to your business requirements .

RULES:

When an error occurred in any Flow :

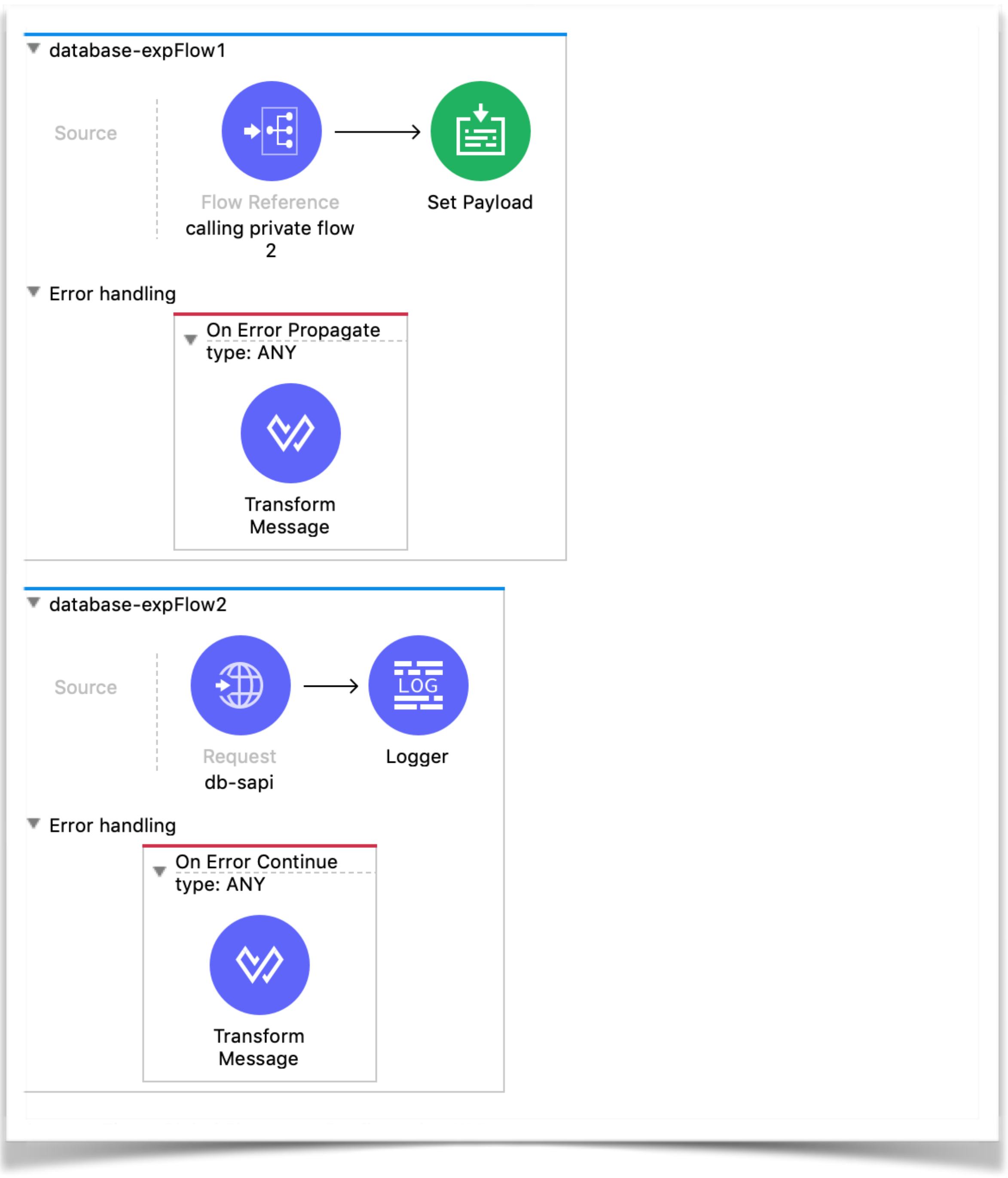
Rule 1 : See if anything is present in Error-Handling . Even if there are on-error-propagate and on-error-continue blocks, see that if that particular errorType is handled. If NOT, then Mule will use default Error Handling . Suppose if your flow is not called by any other flow , then it will display default value that is set in Error-Response and gives status code as 500 bad default if nothing is set manually. If your flow is called by any other flow, then it will RAISE as an error to the calling flow.

Rule 2 : If some Error Handling is present in that particular flow and errorType is handled (YES condition for RULE1) , then check whether it is On-Error-continue or On-Error-Propagate . In both the cases, **Mule will execute all components** within that block.

Rule 3 : After execution of all components . Now:

- If the error is handled using On-Error Propagate , it will raise an error back to the calling flow.
- If the error is handled using On-Error Continue , it will not raise an error back to the calling flow and continue to next processor after "flow-ref" and continues further process as it is. But it will not continue to other processors in the flow where error is handled.
- Suppose you have only single flow. Then On-error-propagate and On-error-Continue behaves same way, instead error-continues gives 200 status and error-propagate gives 500 status. Because as said , even if its error-continue, it will not go to next processor within the same flow.
- **Always remember , this point is very important. On-error-Continue will continue only to next processor of calling flow but not in the same flow .**

If you observe below flows. Suppose your error is handled in 2nd private flow and if there is on-error-continue, it will not execute logger in private flow 2, instead it will go to Set payload of private flow 1 which is actually calling private flow 2



If it is on-error-propagate , The error is raised back to the calling flow . And now flow Rule1 to Rule 3 again for that particular calling flow until final process is completed .

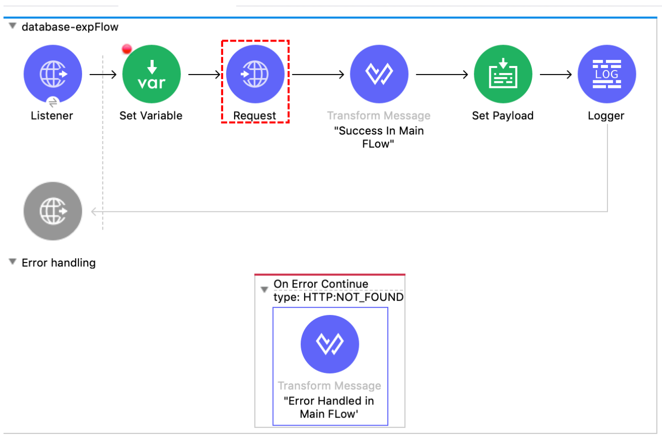
Simple tip is, whenever an error is raised back to calling flow, just go by the rules 1,2,3 again .

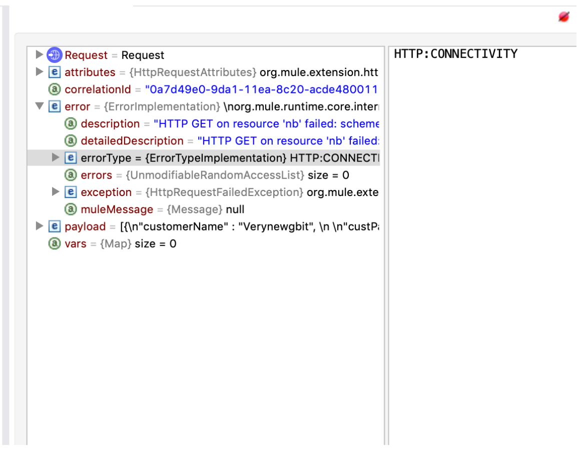
Let's See some examples and check what is the output and status code of each one.

Consider that in all cases , default value for response in Http Listener in Error Response is set to #[payload] and all cases below raise error for errorType : HTTP:CONNECTIVITY

All cases have input payload as {name : "sravan lingam"}

Case 1: Let us assume we have only one flow. Lets follow rules





```
▶ Request = Request
▶ attributes = {HttpRequestAttributes} org.mule.extension.htt
  correlationId = "0a7d49e0-9da1-11ea-8c20-acde480011
▼ error = {ErrorImplementation} \norg.mule.runtime.core.inter
    description = "HTTP GET on resource 'nb' failed: scheme
    detailedDescription = "HTTP GET on resource 'nb' failed:
  ▶ errorType = {ErrorTypeImplementation} HTTP:CONNECTI
    errors = {UnmodifiableRandomAccessList} size = 0
  ▶ exception = {HttpRequestFailedException} org.mule.exte
    muleMessage = {Message} null
▶ payload = [{\n"customerName" : "Verynewgbit", \n \n"custPa
  vars = {Map} size = 0
```

```
HTTP:CONNECTIVITY
```

Rule1 : There's something handled in Error-Handler. But errorType is not Matching with Handled errorType. So don't care whether it's error-continue or error-propagate. When errorType not matched, it will use default error handler by mule.

Actual errorType = HTTP:CONNECTIVITY

Handled errorType = HTTP:NOT_FOUND

As both doesn't match and this is main flow and is not called by any flow,

It uses Mule's default error-handling.

It will print the payload whatever is coming from input just before Http Request with status code 500

Final Response :

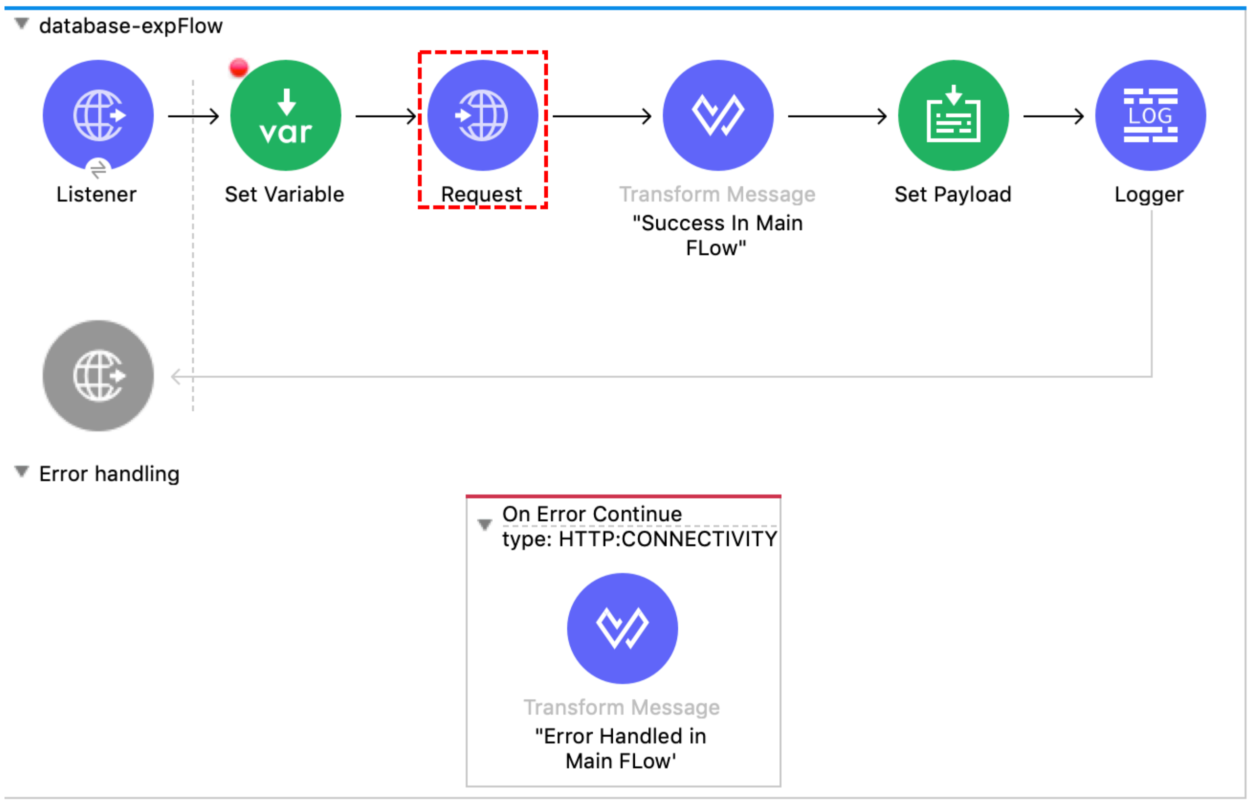
Response Body :

{name : "sravan lingam" }

Response status code :

500

Case 2: Let us assume we have only one flow. Lets follow rules



Rule1 : There's something handled in Error-Handler. And errorType is Matching with Handled errorType.

Actual errorType = HTTP:CONNECTIVITY

Handled errorType = HTTP:CONNECTIVITY

As both are matching and this is main flow and is not called by any flow,

Rule 2 : The flow is using On-Error-Continue . It executes all components in this block. In this block we are setting payload in Transform message with value "Error Handled in Main flow"

Rule 3: According to point b and c in Rule 3, As it is On-error-continue , and the current flow is not called by any other flow , it will execute processors in Error-Continue and end the process with 200 status

It will not execute the components after Http-Request as this is main flow

It will print the payload whatever is set in Transform Message in Error-Continue

Final Response :

Response Body :

“Error Handled in Main flow”

Response status code :

200

Case 3: If we use On-error-Propagate in Case 2 instead of On-error-Continue ,Every rule is same but as its On-error-propagate , it will raise and error back to caller with status code 500

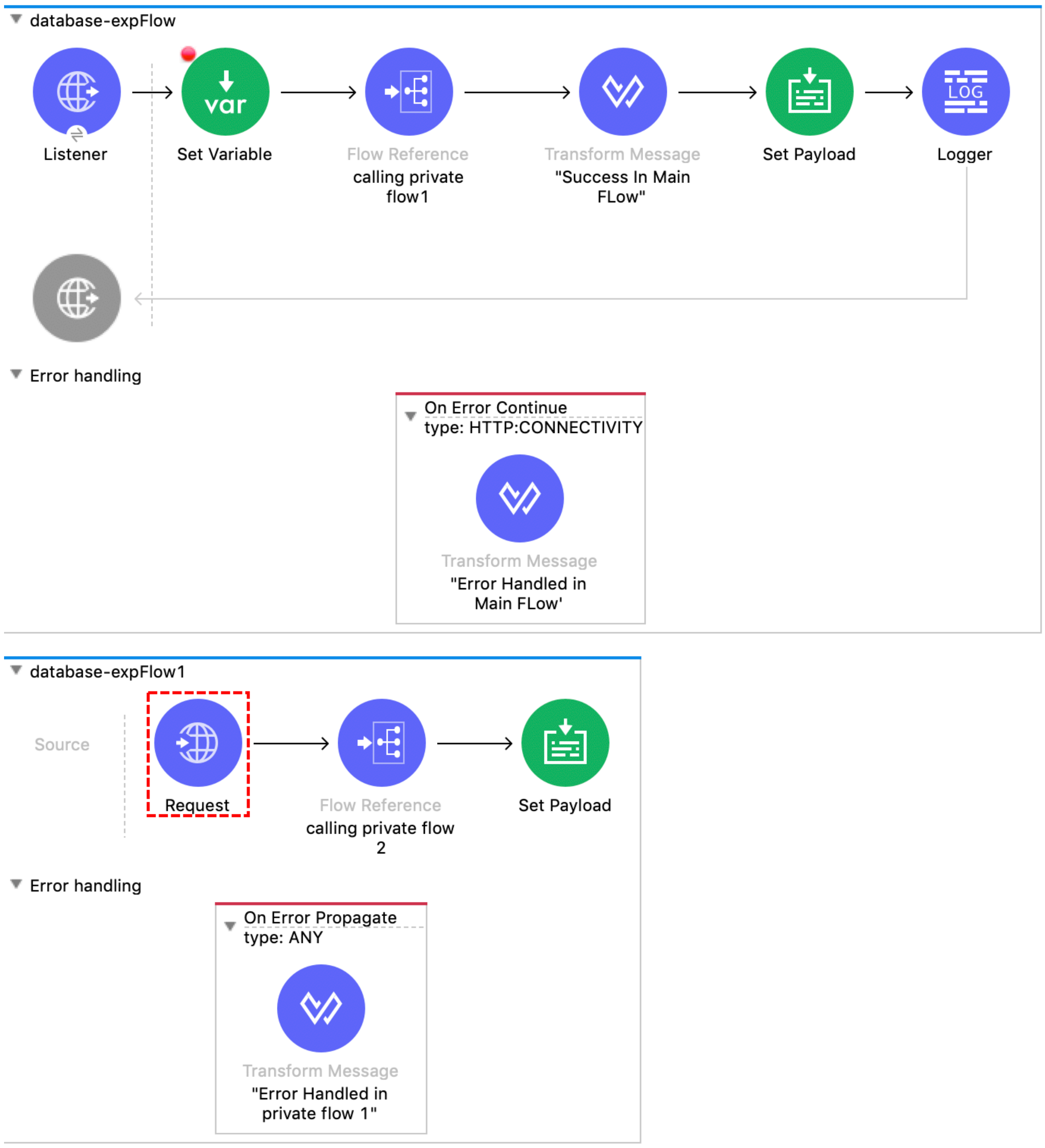
Final Response :

Response Body :

“Error Handled in Main flow”

Response status code :

500



Case 4: Let us assume we have 2 flows. Lets follow rules

Rule1 : There's something handled in Error-Handler. And errorType is Matching with Handled errorType.

Actual errorType = HTTP:CONNECTIVITY

Handled errorType = ANY

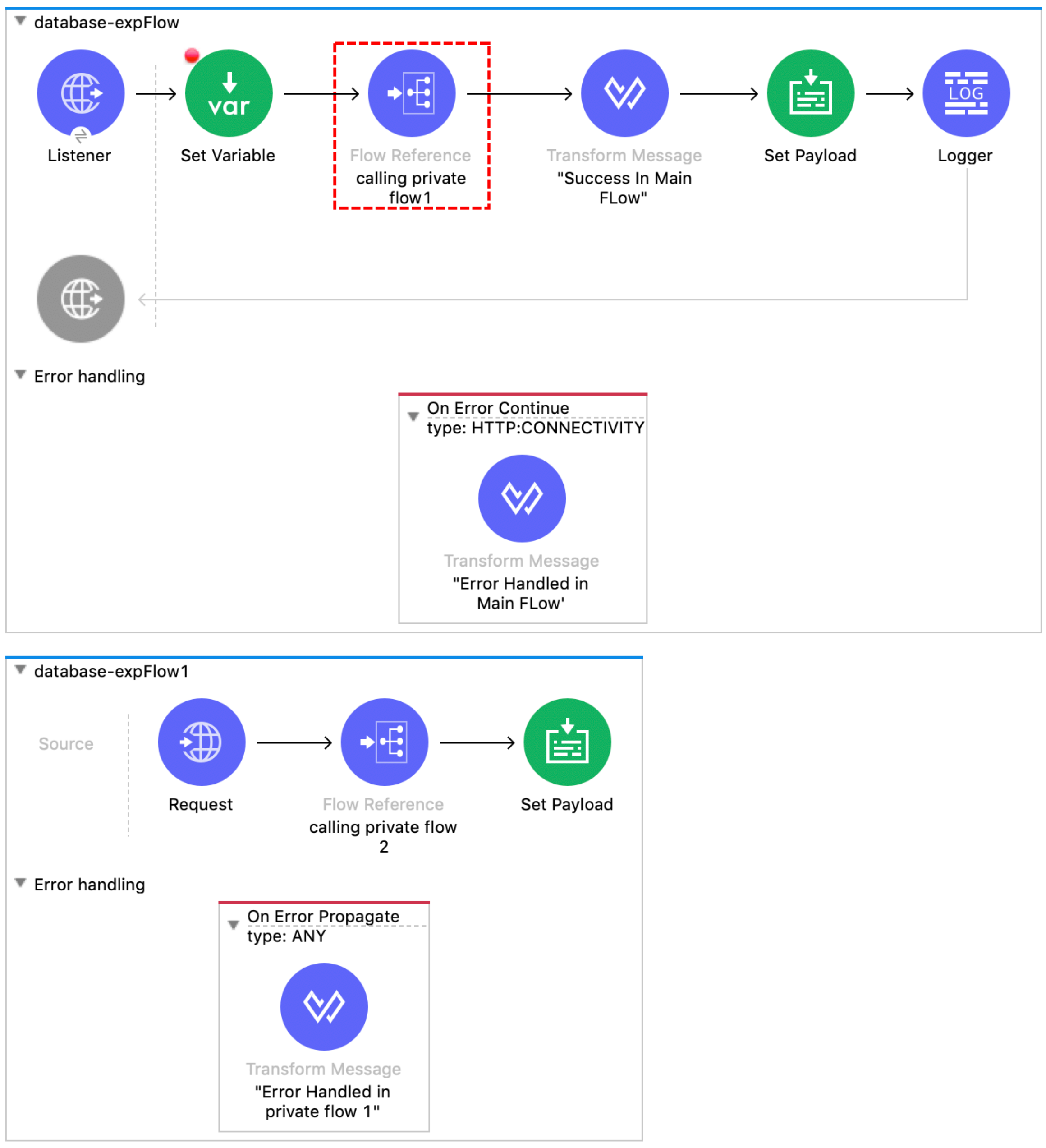
ANY can handle anything

<u>Rule 2 :</u> The flow which has error (private flow 1 in this case) is using On-Error-Propagate . It executes all components in this block. In this block we are setting payload in Transform message with value “Error Handled in Private Flow 1”

<u>Rule 3</u>: According to Rule 3, As it is On-error-propagate , and the current flow is called by main flow , it will execute processors in Error-Propagate and raise error back to calling flow which is main flow.

Now ball is in Main flow court.

Now repeat the rules again for Main flow :



This is how it raised back error to main flow

Repeat rules:

Rule 1 : Matching error type exists

Rule 2 : On-error continue , executes transform message with payload “Error handled in main flow”, will not go to other components after flow-ref. As it is On-error Continue, and no other flow is calling the current flow, it will display latest payload with 200 status

Final Response :

Response Body :

“Error Handled in Main flow”

Response status code :

200

Case 5: Same as Case 4 except that error-continue is used in private flow 1

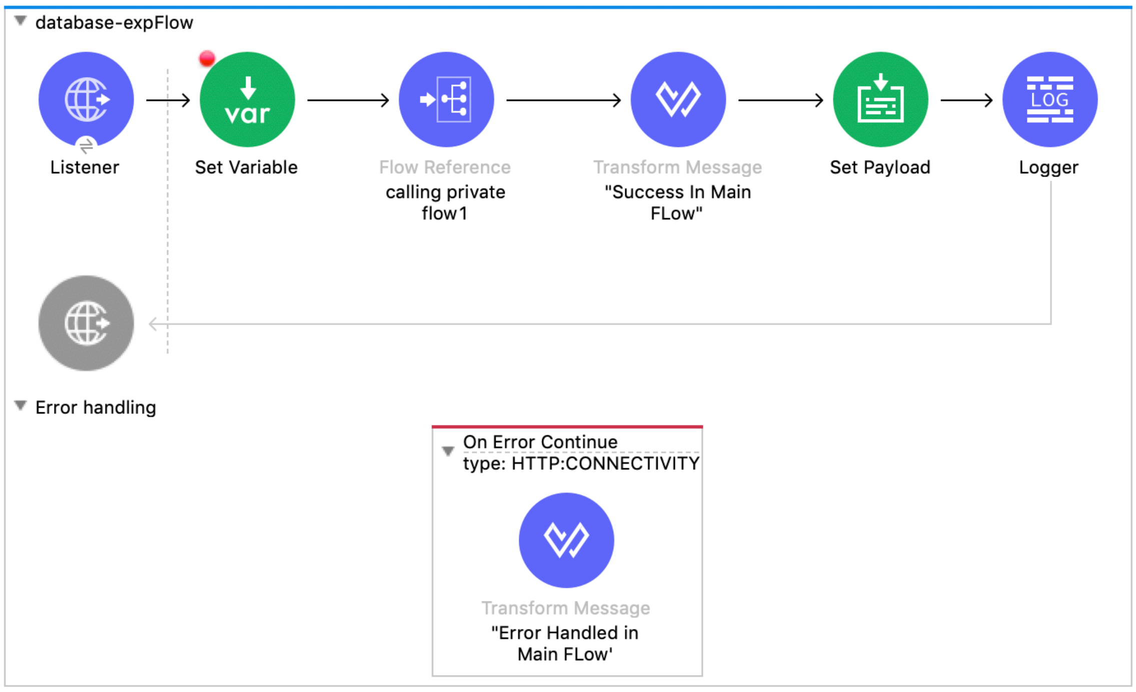
Rule1 : There’s something handled in Error-Handler. And errorType is Matching with Handled errorType.

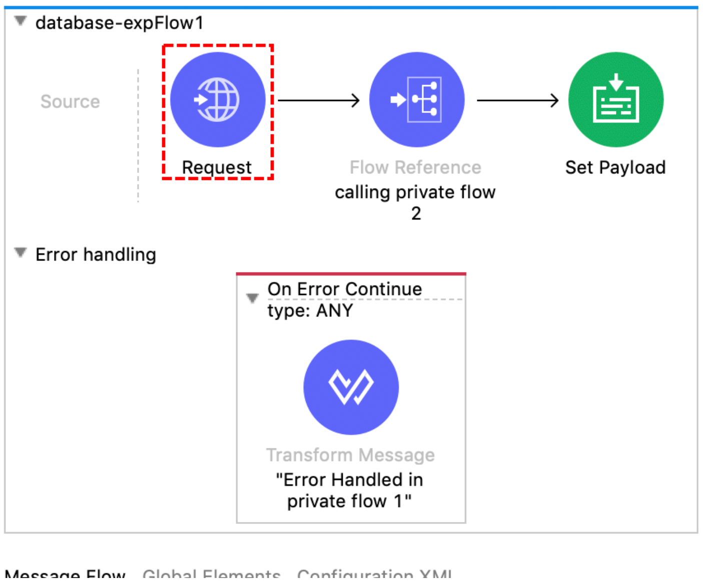
Actual errorType = HTTP:CONNECTIVITY

Handled errorType = ANY

ANY can handle anything

Rule 2 : The flow which has error (private flow 1 in this case) is using On-Error-Continue. It executes all components in this block. In this block we are setting payload in Transform message with value “Error Handled in Private Flow 1”





<u>Rule 3</u>: According to Rule 3, As it is On-error-continue , and the current flow is called by main flow , it will execute processors in Error-continue ,doesn't raise as error and go back to calling flow which is main flow . It executes all components after flow-ref and final payload will be displayed with 200 .

<u>Final Response :</u>

Response Body :

"Success in Main flow"

Response status code : 200

Configuring Properties :

It is always a best practice to place all the values which are hardcoded in the Application in property files. This helps us by not touching our code if there are any changes in those values. We can just modify them in property files.

Also usually we have different configuration details for different environments for connecting to external systems. For eg : We have different host, password for Dev , Int , Stage and Prod environments . In that case we can have 3 property files for each environment which can handle the values based on the environment we deploy our application in.

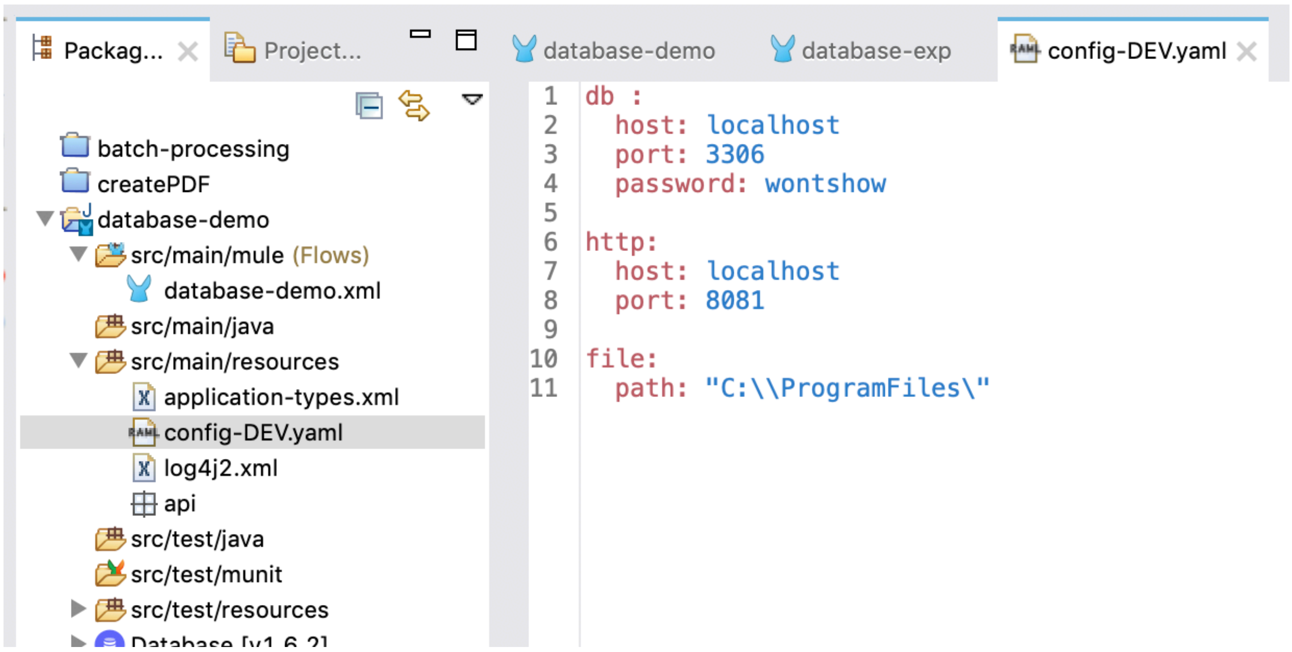
It's very simple. Have your key-value pairs in a property file and use them across your application using ${keyname} or if you want to use in some MEL , you p('keyname').

Supported Files:

The Configuration Properties support both YAML configuration files (.yaml) and Properties configuration files (.properties). String is the only supported type in both file formats. Use the format that you find easier to read and edit.

Remember it's alway best practice to place these files in src/main/resources

YAML Sample :

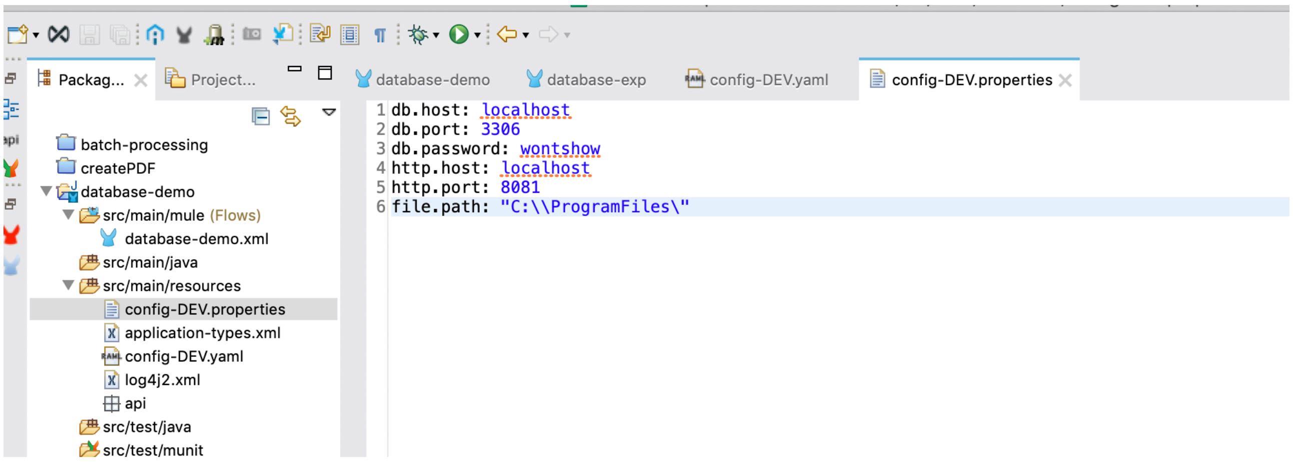


```
db :
  host: localhost
  port: 3306
  password: wontshow

http:
  host: localhost
  port: 8081

file:
  path: "C:\\ProgramFiles\"
```

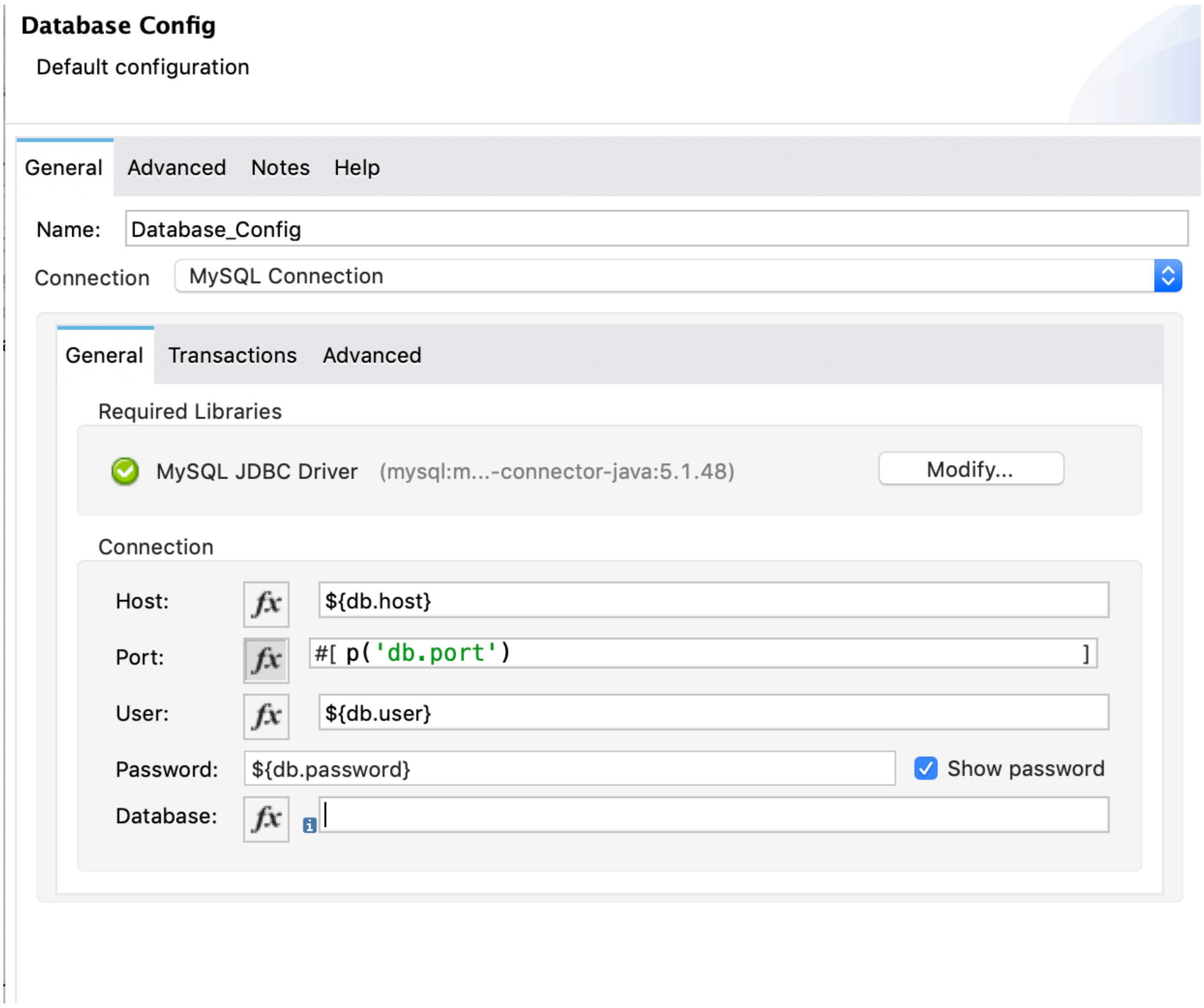
Properties Sample :



```
db.host: localhost
db.port: 3306
db.password: wontshow
http.host: localhost
http.port: 8081
file.path: "C:\\ProgramFiles\"
```

In both cases, we call these property values using ${keyname} or p('keyname')

Eg:



Database Config
Default configuration

General | Advanced | Notes | Help

Name: Database_Config
Connection: MySQL Connection

General | Transactions | Advanced

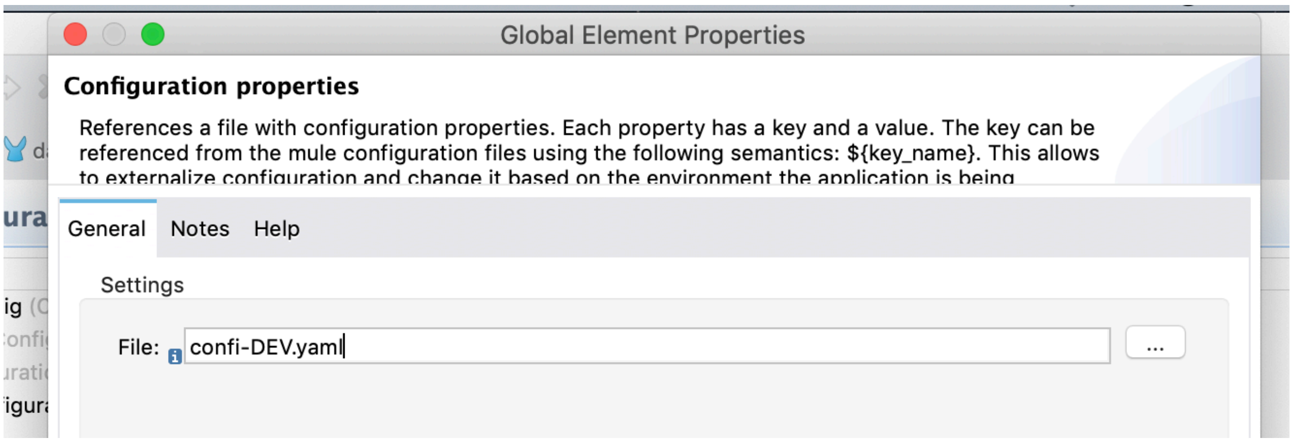
Required Libraries
MySQL JDBC Driver (mysql:m...-connector-java:5.1.48) Modify...

Connection
Host: ${db.host}
Port: #[p('db.port')]
User: ${db.user}
Password: ${db.password} [✓ Show password]
Database:

If you observe in above pic , we use p('db.port') only if we have placeholder for MEL.

(wherever you see fx) . Or else you can you any way to access values.

You can access these values only if you have "Configuration Properties" configured.



Configuration properties

References a file with configuration properties. Each property has a key and a value. The key can be referenced from the mule configuration files using the following semantics: ${key_name}. This allows to externalize configuration and change it based on the environment the application is being

Settings

File: confi-DEV.yaml

You can go to Global-elements —> Create —> search for Configuration properties

Or

Use below xml tag in mule config file:

```
<configuration-properties doc:name="Configuration properties"
doc:id="96988409-1412-4e11-a697-3aa1f600d923" file="config-DEV.yaml" />
```

Note: If you miss any property value in property file and if such property key is accessed in Mule App, then the application fails while deploying saying that so and so property value is missing .

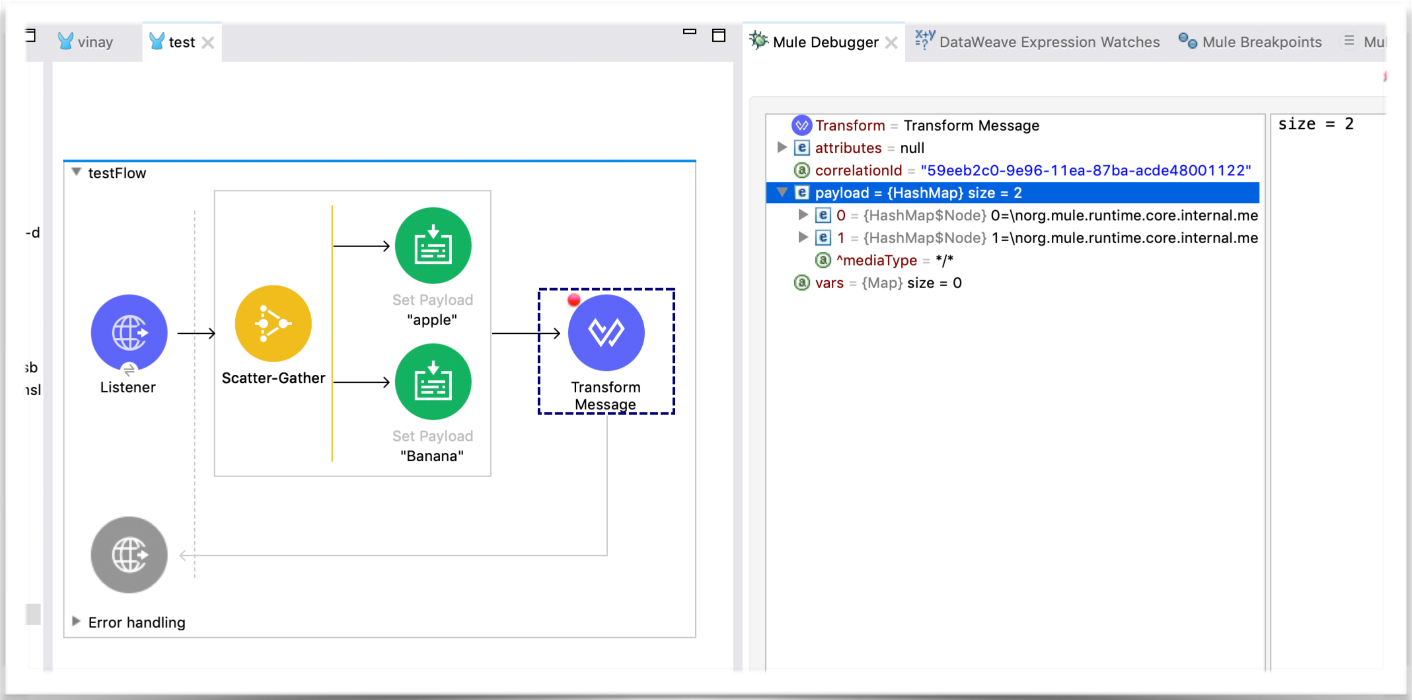
● Scatter - Gather :

Scatter Gather sends the request message to multiple targets concurrently , it collects responses from various targets and aggregate them to single message.

■ The output type Scatter Gather is Object. application/java

■ It executes routes concurrently instead of sequentially . But wait till all targets executes

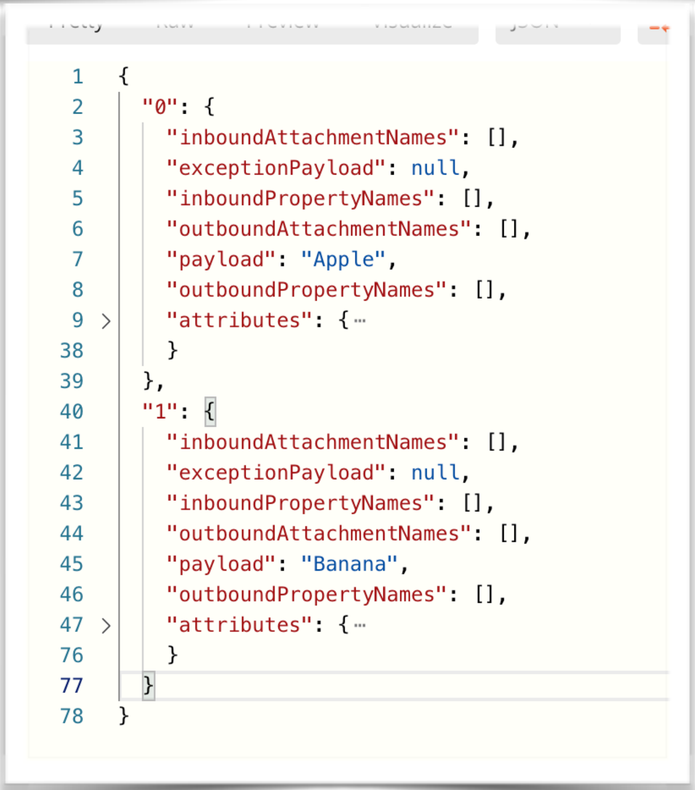
■ Output payload is combination of all targets. And each target output has payload+attributes.



```
Transform = Transform Message
attributes = null
correlationId = "59eeb2c0-9e96-11ea-87ba-acde48001122"
payload = {HashMap} size = 2
  0 = {HashMap$Node} 0=\norg.mule.runtime.core.internal.me
  1 = {HashMap$Node} 1=\norg.mule.runtime.core.internal.me
  ^mediaType = */*
vars = {Map} size = 0
```

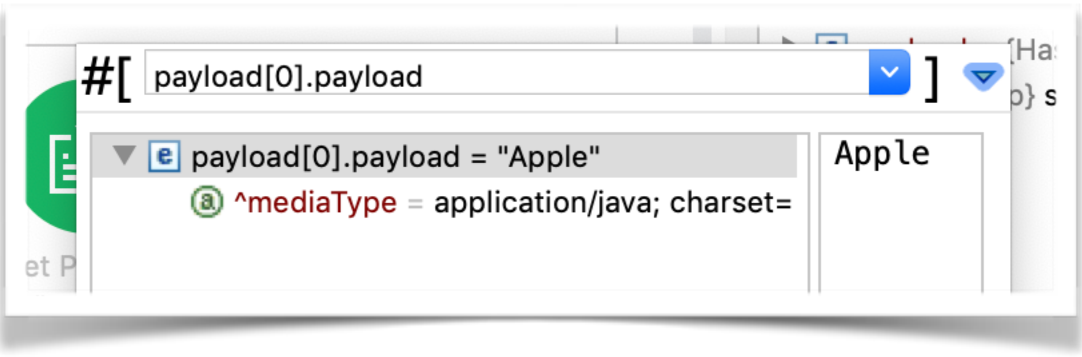
```
size = 2
```

Result —



```
{
  "0": {
    "inboundAttachmentNames": [],
    "exceptionPayload": null,
    "inboundPropertyNames": [],
    "outboundAttachmentNames": [],
    "payload": "Apple",
    "outboundPropertyNames": [],
    "attributes": {…
    }
  },
  "1": {
    "inboundAttachmentNames": [],
    "exceptionPayload": null,
    "inboundPropertyNames": [],
    "outboundAttachmentNames": [],
    "payload": "Banana",
    "outboundPropertyNames": [],
    "attributes": {…
    }
  }
}
```

If you want to access only payload value, access like below:



```
#[ payload[0].payload ]
payload[0].payload = "Apple"
    ^mediaType = application/java; charset=
Apple
```

Note: If any of the targets fail inside scatter-gather , it will fail whole process.

So its always best practice to wrap every target within Try-Block with On-Error-Continue ,So that you can handle the errors and complete the process successfully .

● Validation :

Validation component will verify the content of message whether it matches the specified criteria.

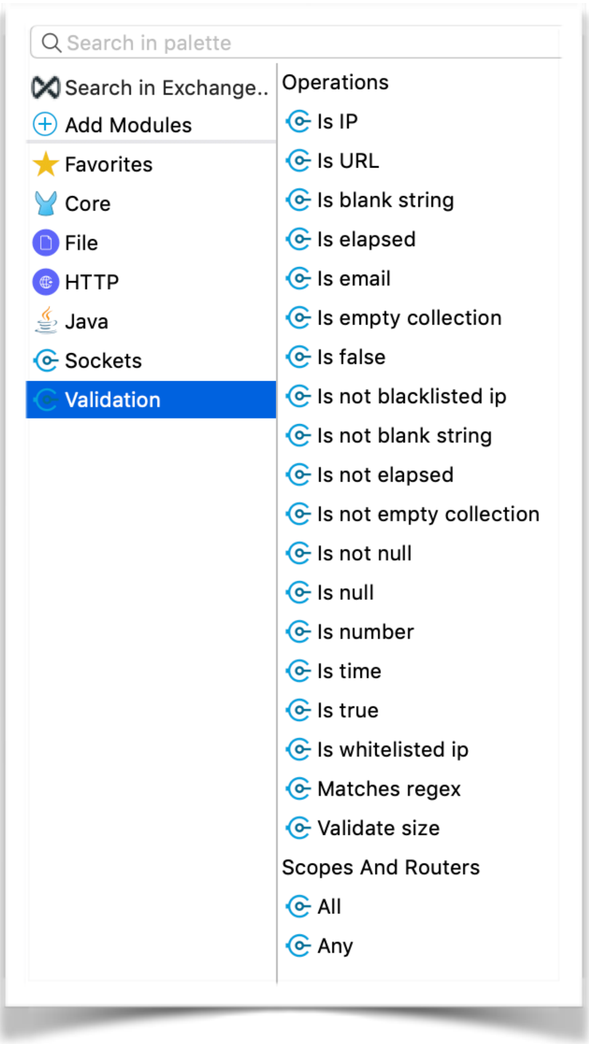
Examples:

```
<validation:is-not-blank-string value="#[payload]" message="The username cannot be blank"/>
```

```
<validation:is-not-null value="#[payload]" message="Null is not a valid value"/>
```

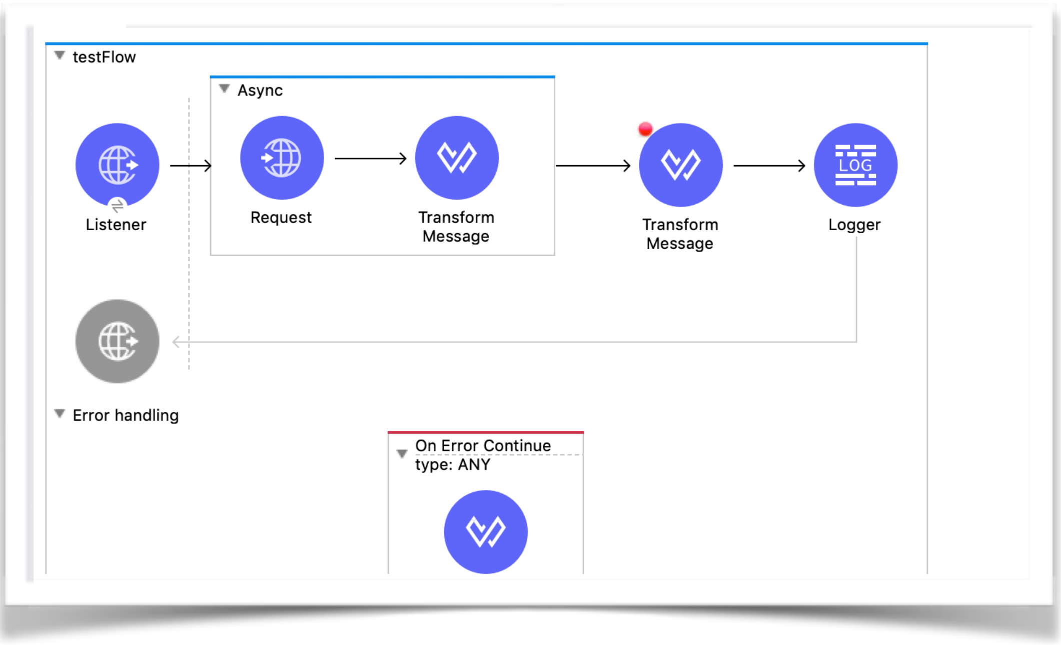
```
<validation:is-true expression="#[vars.username == "sravan"]"/>
```

It will proceed to further connector if the conditions are matched. Else , it will raise an error.



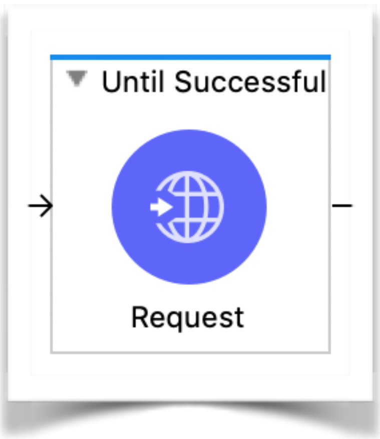
● Async :

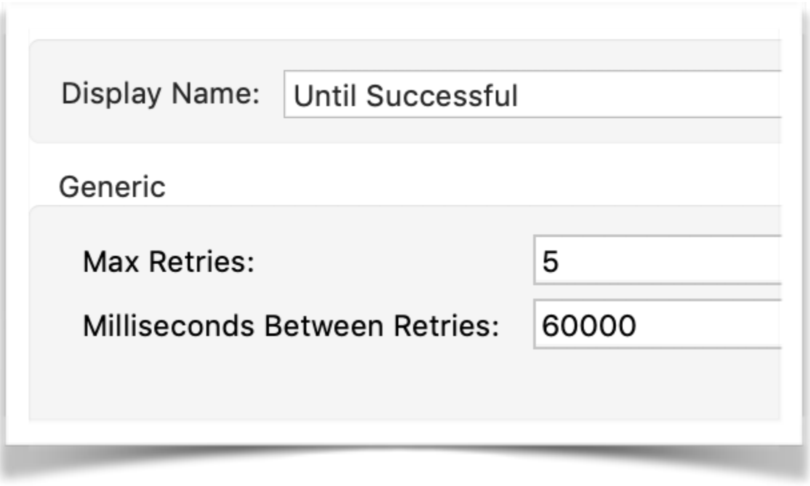
Async block process the request simultaneously with the main flow instead of waiting for the process within Async block to complete.
Usually Async will be used for time-consuming operations. Async scope doesn't Inherit exception strategy of main flow.



● Until-successful :

If a component within the Until-successful fails to connect or produce a successful result, Until Successful retries the failed task until all configured retries are exhausted. If a retry succeeds, the scope proceeds to the next component. If the final retry does not succeed, Until Successful produces an error. The error Type for this Until-successful is MULE:RETRY_EXHAUSTED .



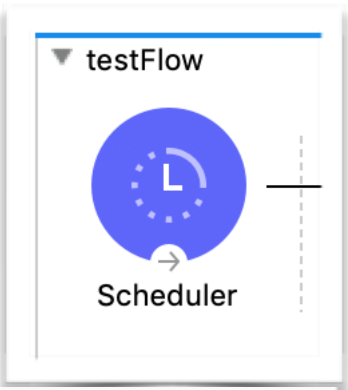


Display Name:	Until Successful

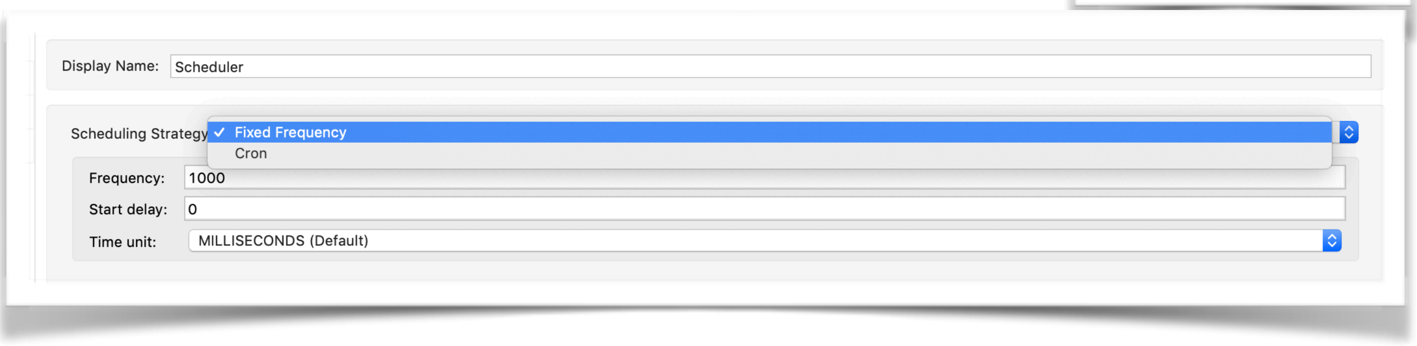
Generic

Max Retries:	5
Milliseconds Between Retries:	60000

● Scheduler:



The Scheduler component enables you to trigger a flow when a time-based condition is met. You can configure it to be triggered at a regular interval like to start a flow every 5 seconds or give it a more flexible cron expression like your flow must start at 2PM IST during weekdays
Scheduler Should be placed only Source part of the flow.



Display Name:	Scheduler
Scheduling Strategy	✓ Fixed Frequency / Cron
Frequency:	1000
Start delay:	0
Time unit:	MILLISECONDS (Default)

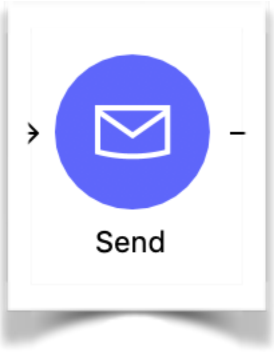
Expression	Behavior
`1/2 * * * * ?`	Run every 2 seconds of the day, every day.
`0 15 10 ? * *`	Run at 10:15 a.m., every day. `0 15 10 * * ? *` and `0 15 10 * * ?` produce the same effect.
`0 15 10 * * ? 2019`	Run at 10:15 a.m., every day during the year 2019.
`0 * 14 * * ?`	Run every minute starting at 2pm and ending at 2:59pm, every day.
`0 0/5 14 * * ?`	Run every 5 minutes starting at 2pm and ending at 2:55pm, every day
`1 1 1 1,7 * ?`	Run the first second of the first minute of the first hour, on the first and seventh day, every month.

Note that the Scheduler component also supports Quartz Scheduler special characters:

- *: All values.
- ?: No specific value.
- -: Range of values, for example, 1-3.
- ,: Additional values, for example, 1,7.
- /: Incremental values, for example, 1/7.
- L: Last day of the week or month, or last specific day of the month (such as 6L for the last Saturday of the month).
- W: Weekday, which is valid in the month and day-of-the-week fields.
- #: Nth day of the month. For example, #3 is the third day of the month.

If you are still not sure about cron expressions you can generate them with available online cron generators. **Eg: http://www.cronmaker.com/**

● EMAIL:



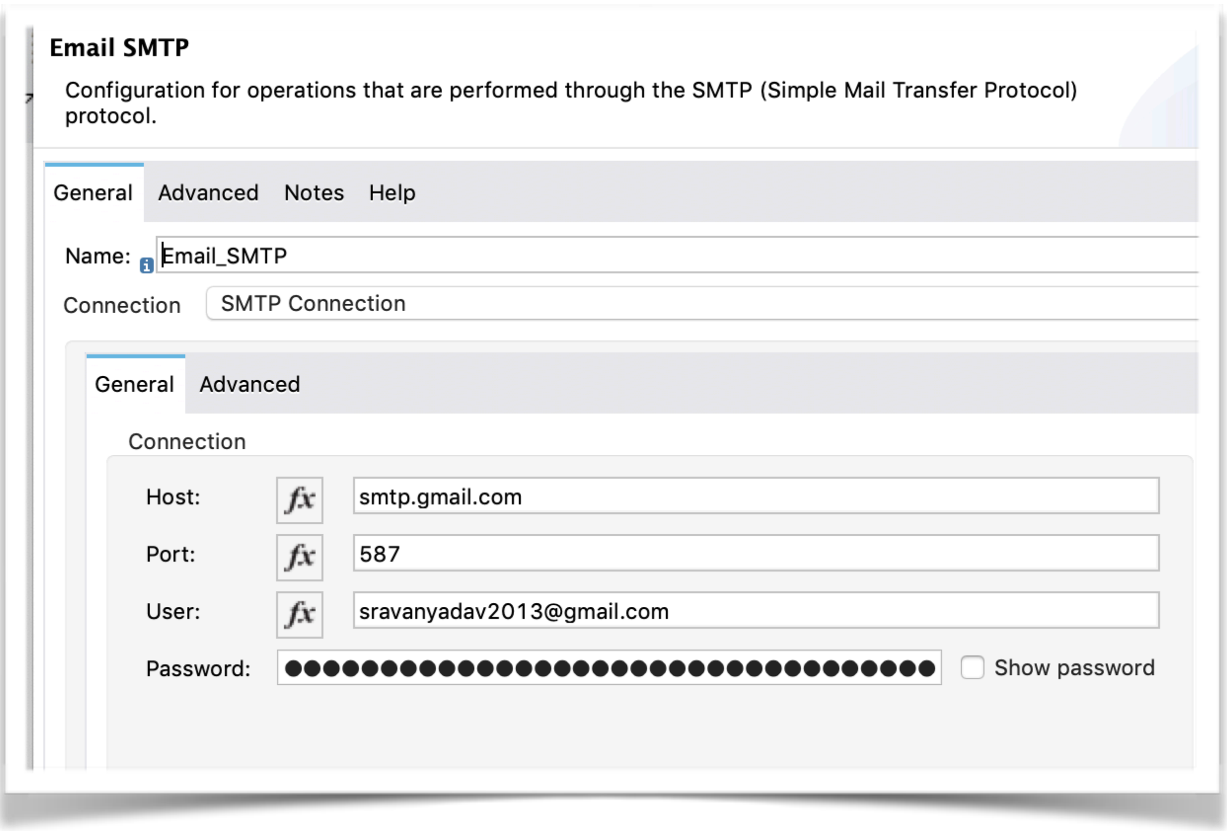
Email connector helps in sending and receiving emails via standard email protocols.
It does following operations :

- Retrieving emails from POP3 mailboxes
- Retrieving, deleting, and storing emails from IMAP mailboxes
- Sending emails over the SMTP protocol
- Supporting secure connections for all protocols over Transport Layer Security (TLS)

Eg: If you want to send a mail via GMAIL, refer to following steps:

Step 1 : Select SMTP connection
Step 2 : Give the configuration details as below

host = smtp.gmail.com
Port = 587
user = your gmail id
password = gmail account password



Email SMTP

Configuration for operations that are performed through the SMTP (Simple Mail Transfer Protocol) protocol.

General | Advanced | Notes | Help

Name: Email_SMTP
Connection: SMTP Connection

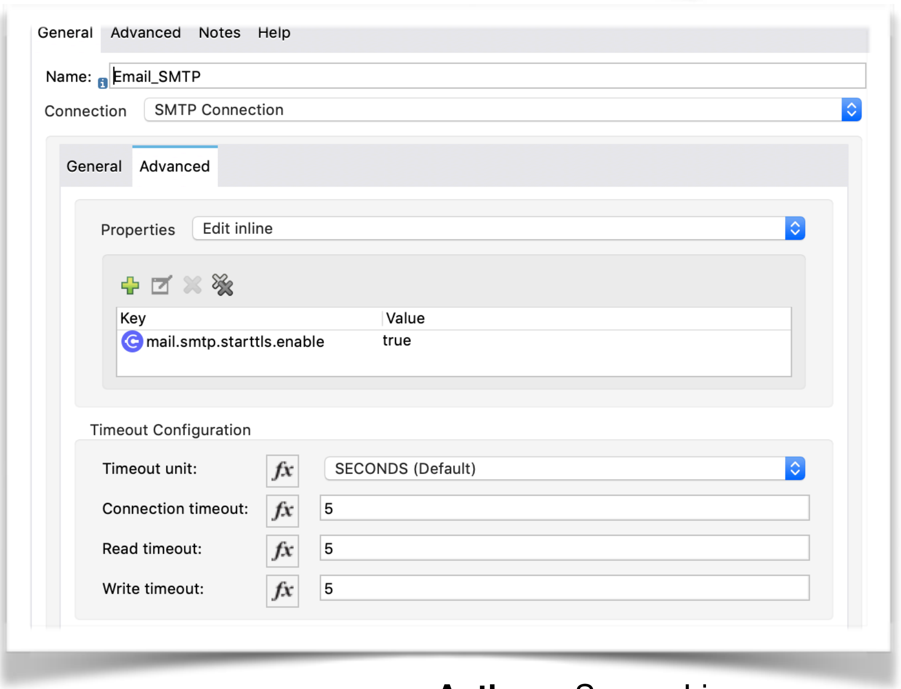
General | Advanced

Connection

Host: smtp.gmail.com
Port: 587
User: sravanyadav2013@gmail.com
Password: ●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●● ☐ Show password

Step 3: Additionally add the property in Advanced Tab

mail.smtp.starttls.enable : true



General | Advanced | Notes | Help

Name: Email_SMTP
Connection: SMTP Connection

General | Advanced

Properties: Edit inline

Key	Value
mail.smtp.starttls.enable	true

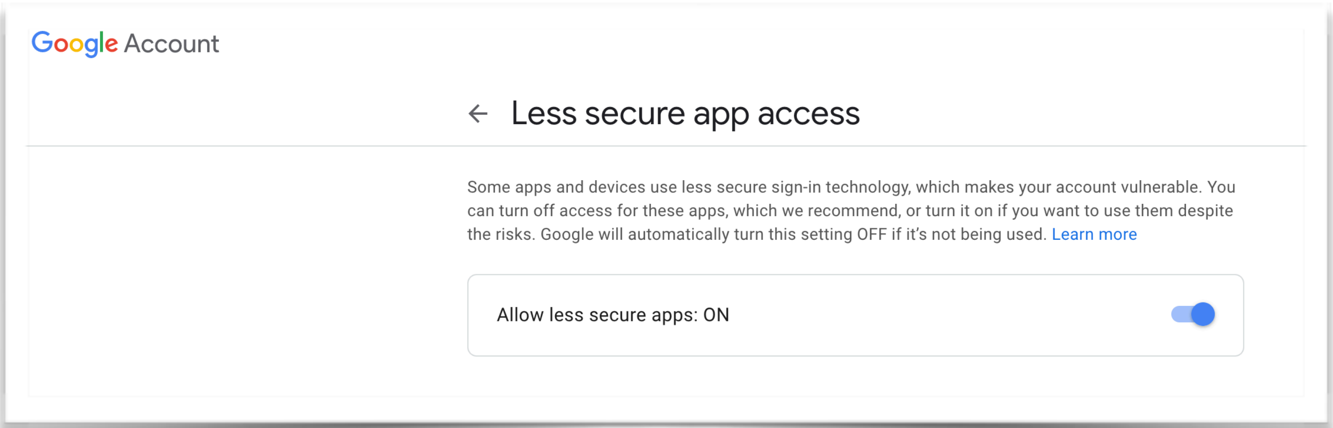
Timeout Configuration

Timeout unit: SECONDS (Default)
Connection timeout: 5
Read timeout: 5
Write timeout: 5

Step 4: Go to your Google Settings for lessSecureApps and enable it . If we don't do this , we won't be able to send a mail via external sources.

Note : This is not valid for Gmail accounts using 2-way authentication

https://myaccount.google.com/lesssecureapps?pli=1

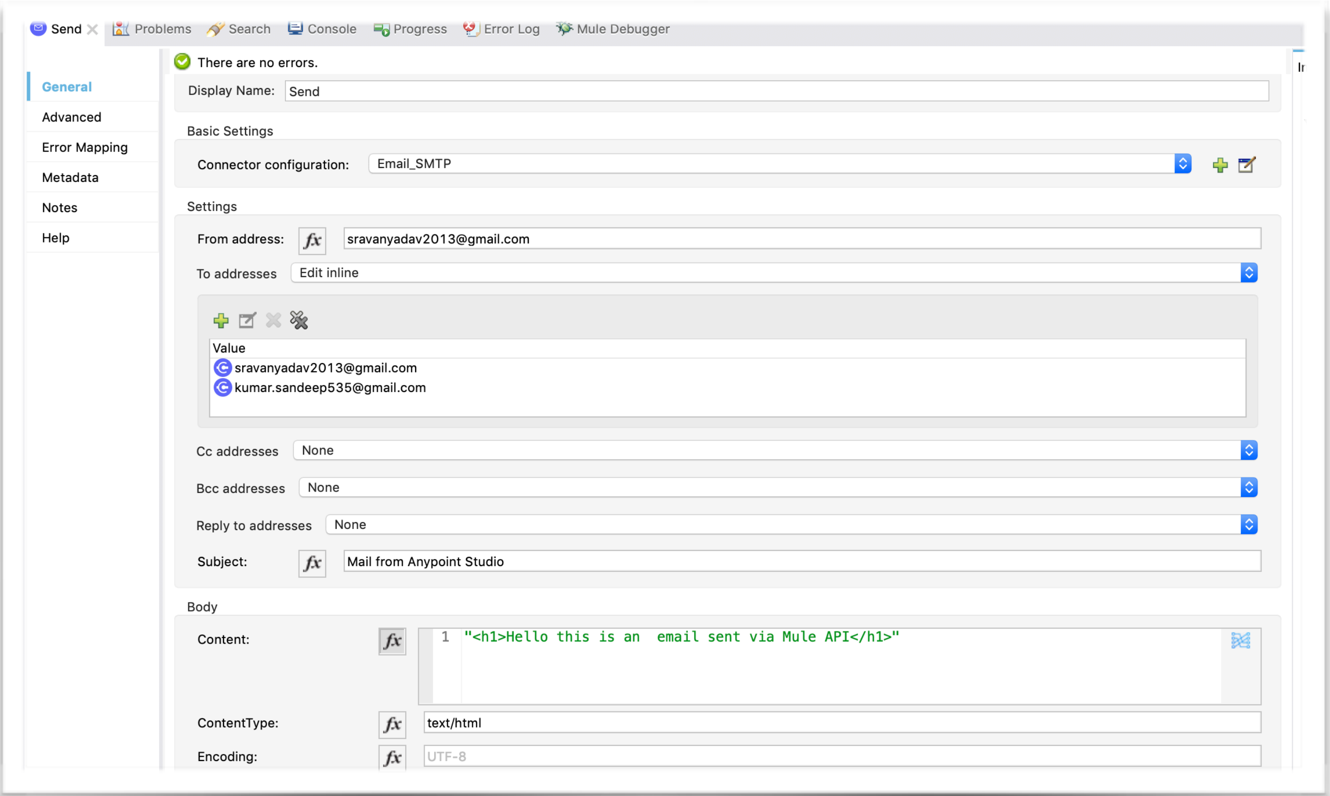

Google Account

← Less secure app access

Some apps and devices use less secure sign-in technology, which makes your account vulnerable. You can turn off access for these apps, which we recommend, or turn it on if you want to use them despite the risks. Google will automatically turn this setting OFF if it's not being used. Learn more

Allow less secure apps: ON

Step 5: Now, final step is to give From address ,to address ,subject along with content . You can also send Attachment by converting them to base64



There are no errors.

Display Name: Send

Basic Settings

Connector configuration: Email_SMTP

Settings

From address: sravanyadav2013@gmail.com

To addresses: Edit inline

Value
sravanyadav2013@gmail.com
kumar.sandeep535@gmail.com

Cc addresses: None

Bcc addresses: None

Reply to addresses: None

Subject: Mail from Anypoint Studio

Body

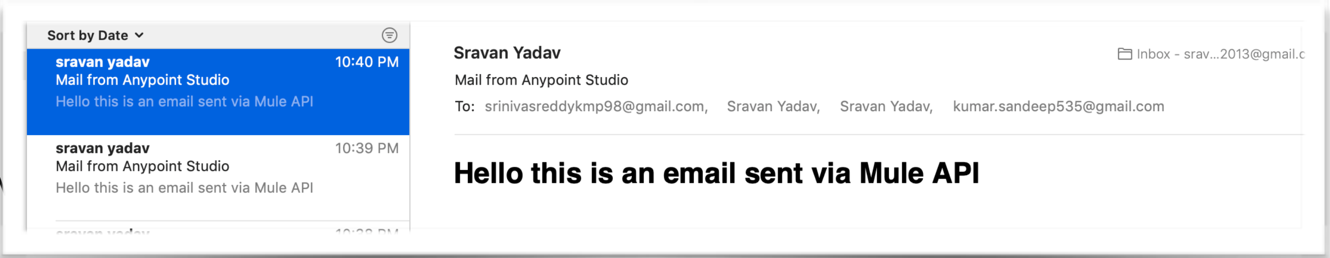
Content:

```
"<h1>Hello this is an  email sent via Mule API</h1>"
```

ContentType: text/html

Encoding: UTF-8

N



sravan yadav 10:40 PM
Mail from Anypoint Studio
Hello this is an email sent via Mule API

sravan yadav 10:39 PM
Mail from Anypoint Studio
Hello this is an email sent via Mule API

Sravan Yadav
Inbox - srav...2013@gmail.c
Mail from Anypoint Studio
To: srinivasreddykmp98@gmail.com, Sravan Yadav, Sravan Yadav, kumar.sandeep535@gmail.com

Hello this is an email sent via Mule API

MUNITS

As we all know that MUnit is a testing framework which helps us to build automated Tests for our API's.
In short , MUnit's helps us to cross check whether our flows are running as expected!
When we say as expected , it includes both Success and Failure scenarios.
This article helps you to create the Test cases and give an idea on how to Mock The Processors and build and run test cases successfully.
Before getting started , lets see what all versions i have used to create and Run these MUnits:

- Anypoint Studio - 7.4 version
- Mule Runtime - 4.2.2 version
- MUnit Version - 2.2.1 version

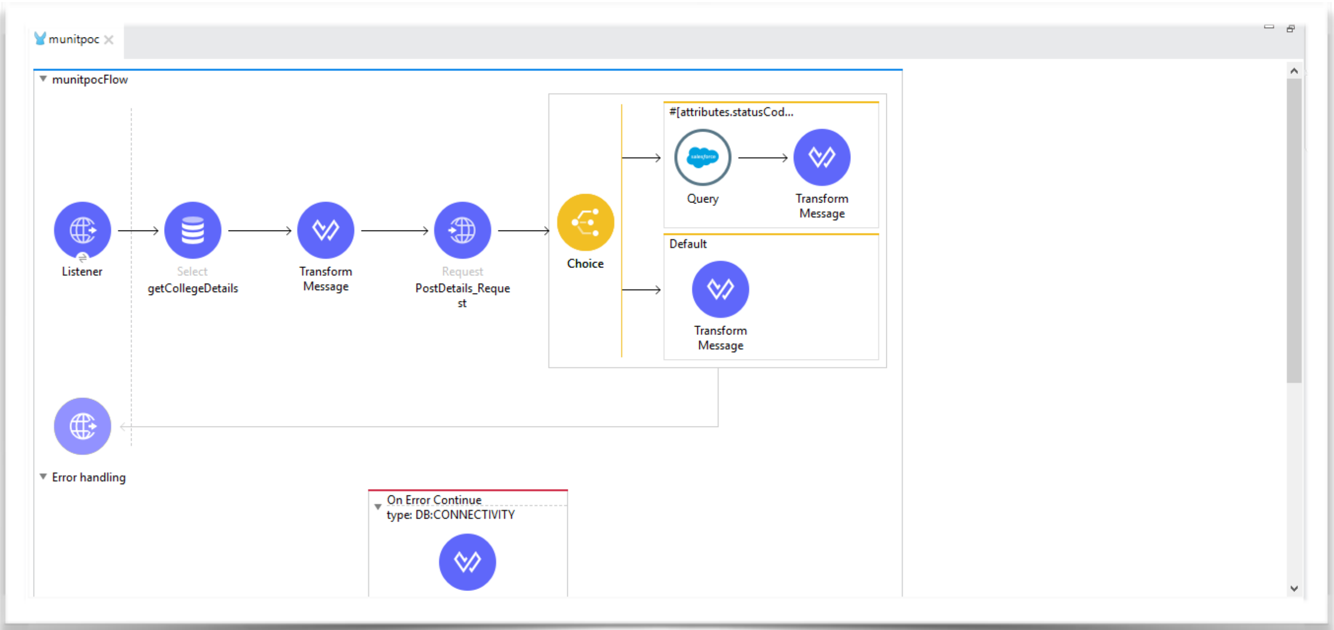
Let's go step-by-step :

1) Configuring pom.xml :

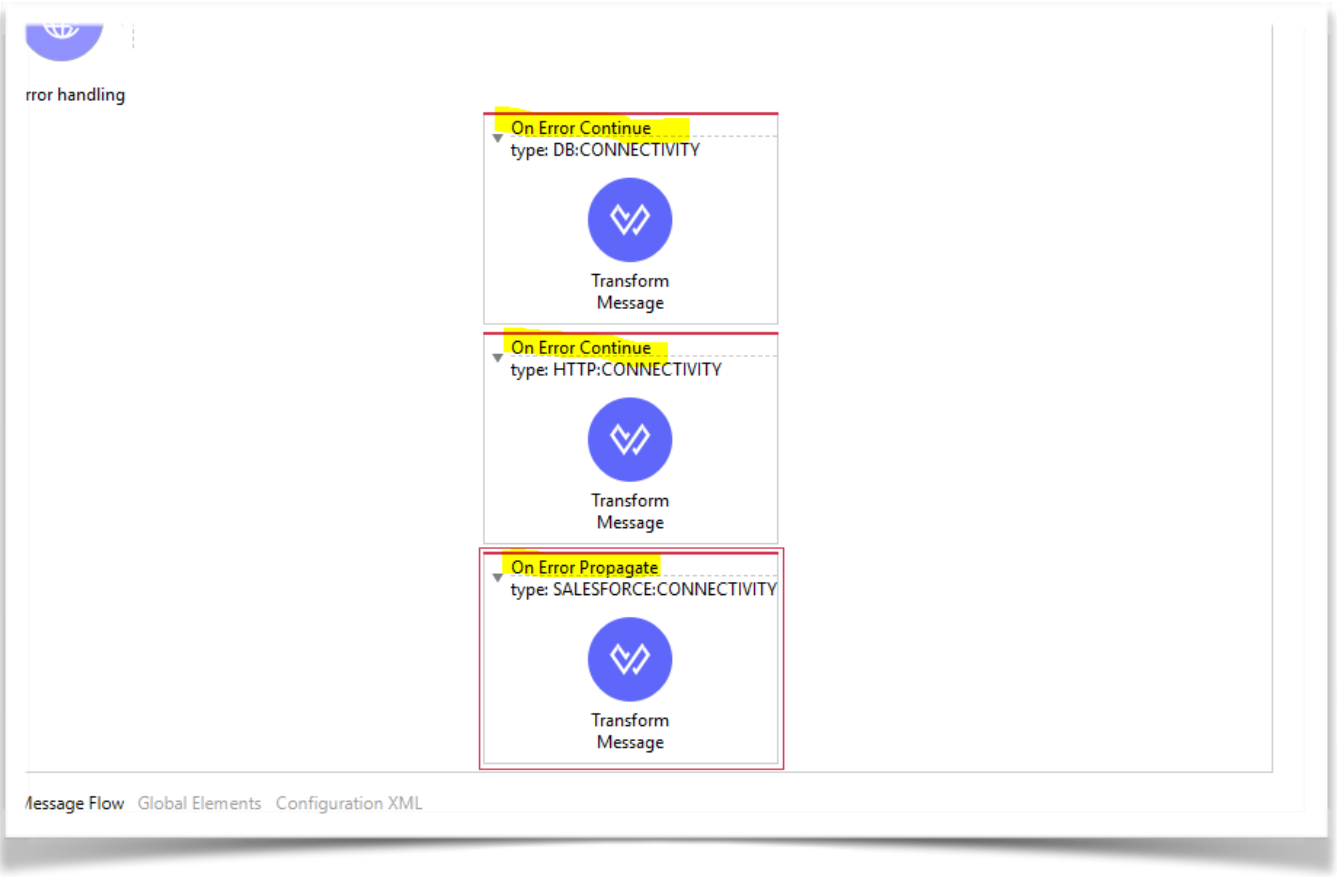
Whenever you right click on the project--> New -> MUnit Test, the dependencies, <plugin > are automatically loaded into pom. But still have a cross check if you have following things in plate before creating MUnits. Check below pom if you have everything as expected.

2) Check the processors which needs Mocking :

Lets have a Mule flow where you have a functionality which includes Database, Http Requester and Salesforce connectors embedded in it!



Also let us have the errors handled. In this scenario we are using both Error Continue & Error Propagate so that i will show you why and how we need to handle MUnits separately for both Error scenarios.



What is Mocking?

Mocking, the word itself describes that we need to Mock the processor with some dummy payload . Simply , we are not actually connecting to the external systems. We are trying to show that "this would be the payload/attribute" when we actually connect. So we would be setting sample or dummy payload for that particular processor without actually connecting to it.

Now What?

Come on let's check what all external systems that we are connecting to ,in this flow

- Database
- Http Requester
- Salesforce

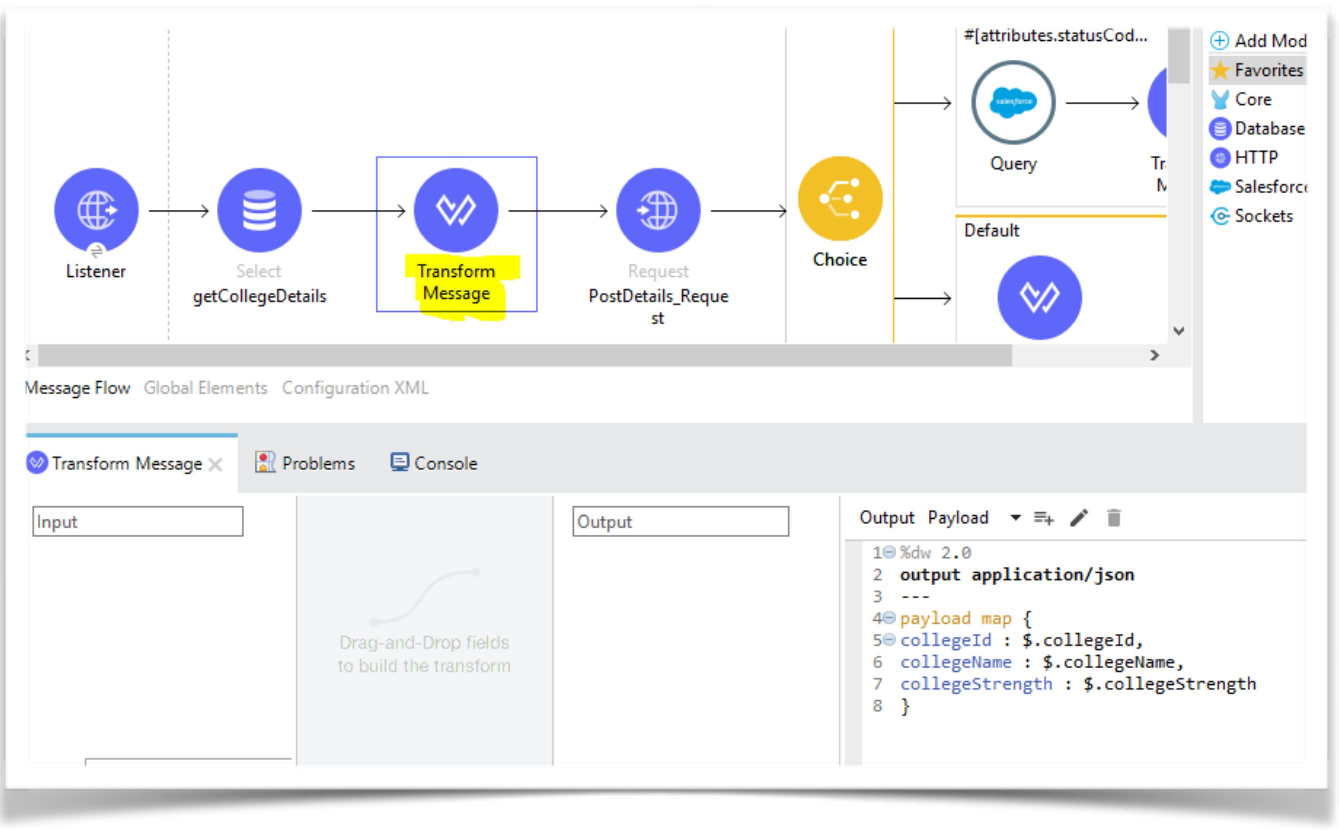
The best practice on how to give Mock payload/attributes is, check what are the processors placed after these external connectors.

In the current flow ,

Lets first check what is there After:

a) Database:

We have a transform Message after Database . And this contains below transformation:



```
%dw 2.0
output application/json
---
payload map {
collegeId : $.collegeId,
collegeName : $.collegeName,
collegeStrength : $.collegeStrength
}
```

We can see that some payload is mapped with collegeId,collegeName,collegeStrength
So Mock the DB with some payload which is an array of data which contains above fields. You can see the Mocking section

b) Http Requester:

We have a choice router after Http requester :
We can see that we are routing based on **attributes.statusCode** .
So Mock the Http requester with attribute containing statusCode . You can see the Mocking section

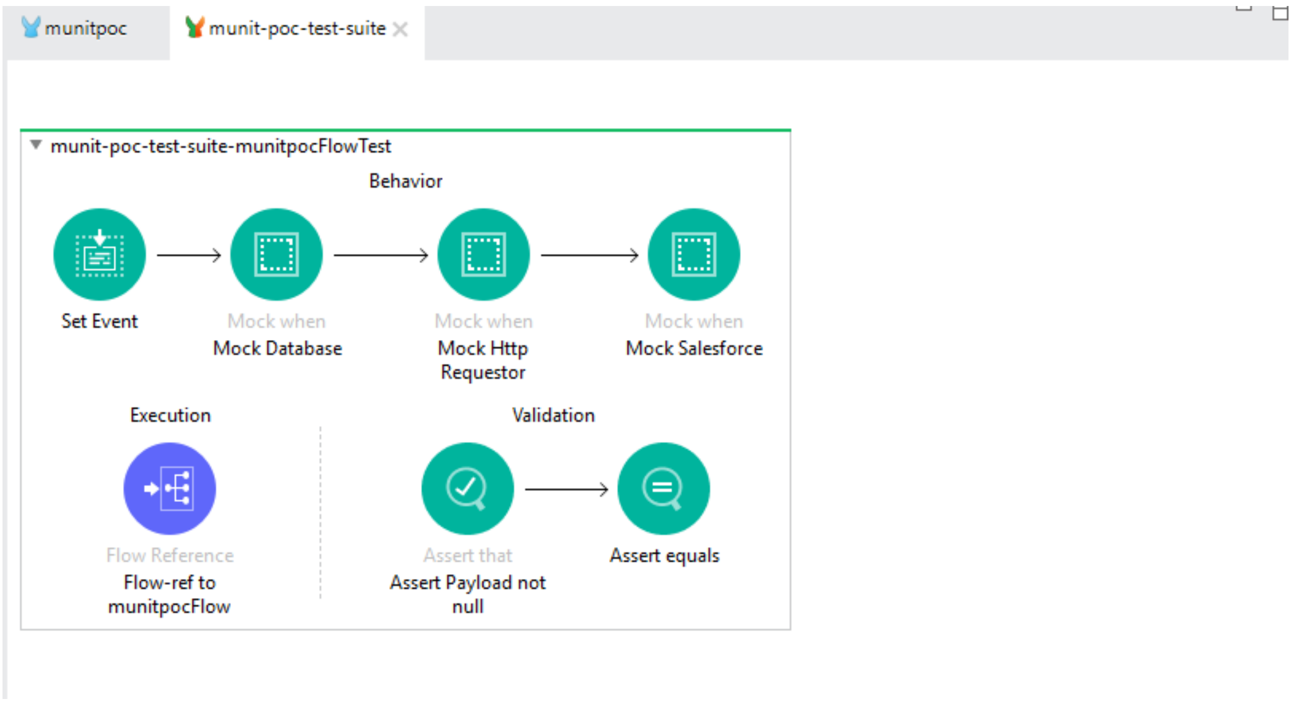
c)Salesforce:

We have a Transform message after Salesforce connector . As of now , we are not making use of any kind of payload that is coming out of Salesforce . But still you need to mock something .because if you are not mocking with some payload or attributes, then your test case will actually try to connect to Salesforce and get data.To make you understand, i have used this scenario . You can see the Mocking section.

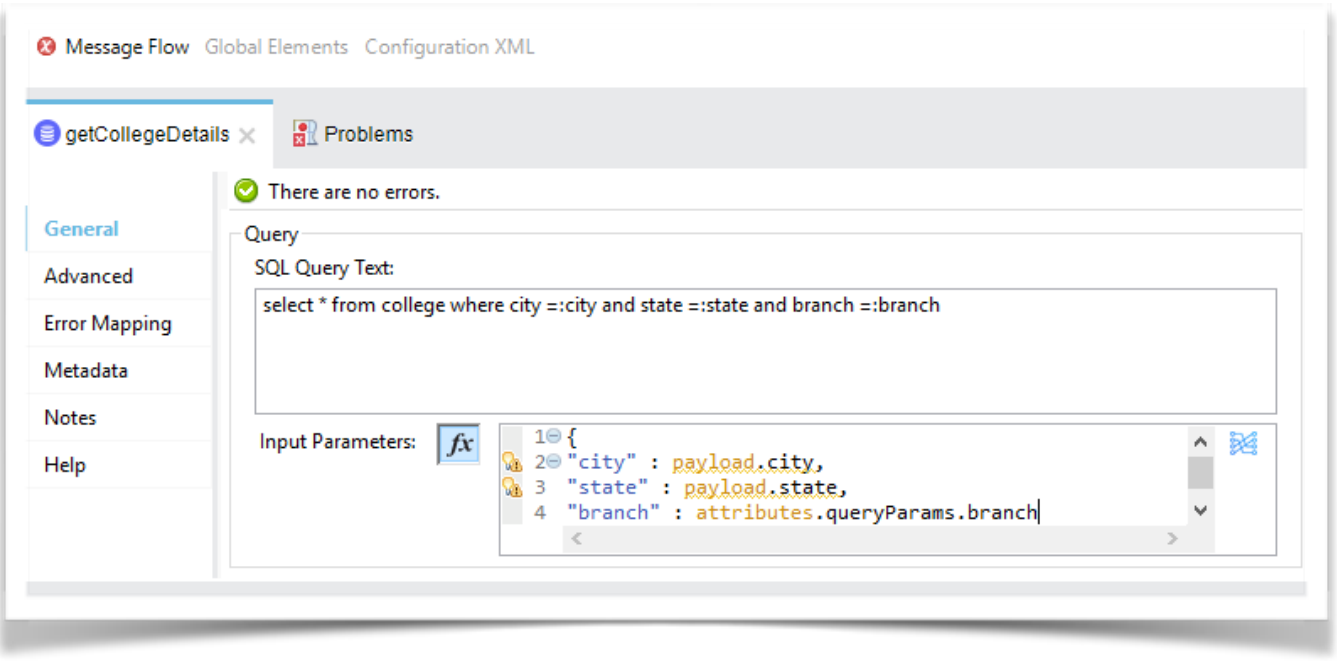
Now are we done?

No. What about error scenarios?
In this flow, we have divided handling of errors for each of the external connectors. We have used Error Continue for DB & Http Connectors and Error Propagate for Salesforce Connector.

I said previously that we are not actually connecting to external systems, then how can we get an error? How to get this test case tested?
There is answer for that!
As we are mocking Payload and attributes , we can also mock the connector to generate an error using **Type Id.**
We shall see how further.
That's all! we are good to write the test cases :)
3) Writing Munit's :
In this project am writing 5 test cases for 100% coverage. Will show you few and rest you can understand from the code am going to share!
a) Writing a Test case for success scenario :
For having a success scenario, keep the things in mind that i mentioned what needs to be mocked in Step 2.



Generally when we run an application , we use postman to run the end-point and send some body if that is a POST method. But Munit's we dont need anything to be run externally using any browser. But still your listener needs get some Data that it is actually expecting. In short, Set Event is something that we pass as how we pass data through postman or any browser.
You can see in Database there is a where clause for the query :



There are no errors.

Query

SQL Query Text:

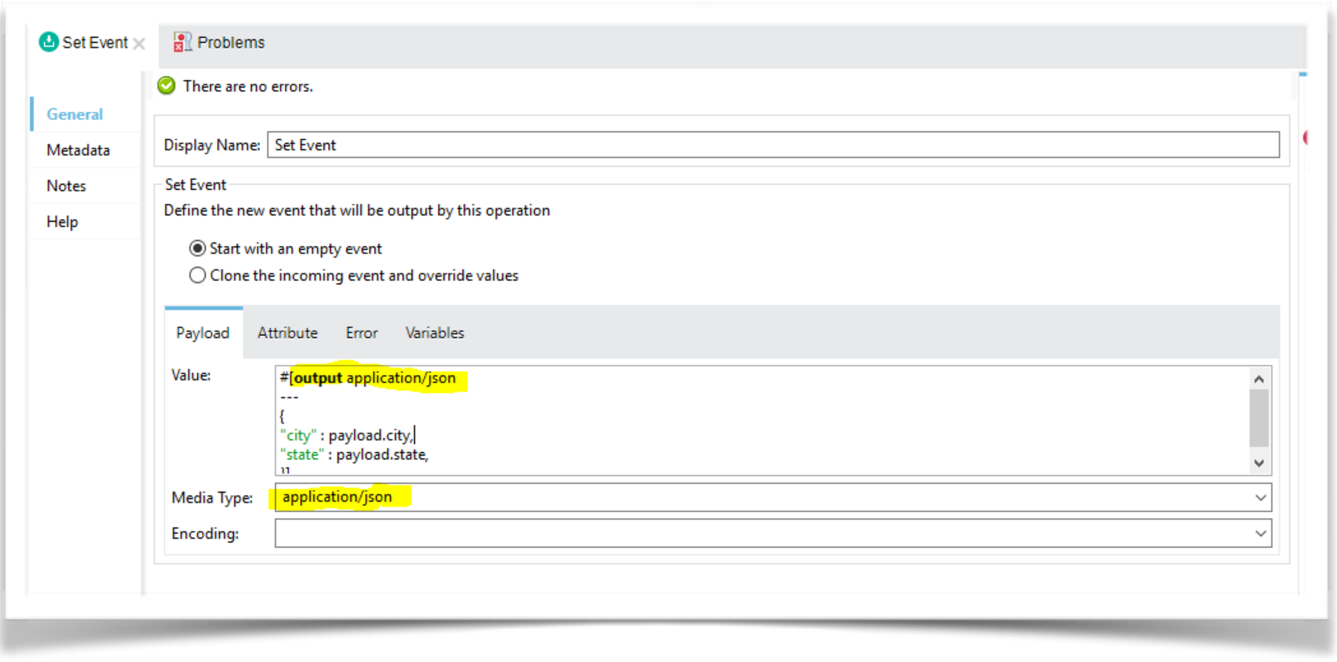
```
select * from college where city =:city and state =:state and branch =:branch
```

Input Parameters:

```
{
"city" : payload.city,
"state" : payload.state,
"branch" : attributes.queryParams.branch
```

so from above image you can see , we need city and state which is coming from payload and branch coming from query parameters.

So we need to set them using **Set Event**



There are no errors.

Display Name: Set Event

Set Event

Define the new event that will be output by this operation

[✓ Start with an empty event]

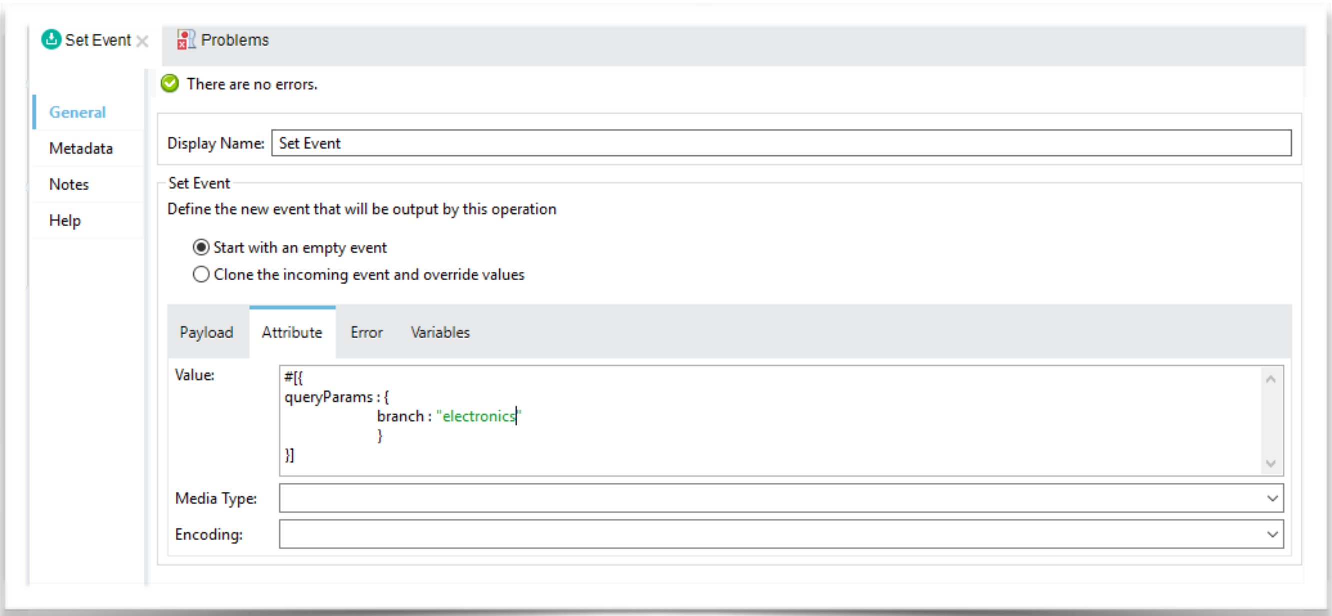
Clone the incoming event and override values

Value:

```
#[output application/json
---
{
"city" : payload.city,
"state" : payload.state,
```

Media Type: application/json

Encoding:



There are no errors.

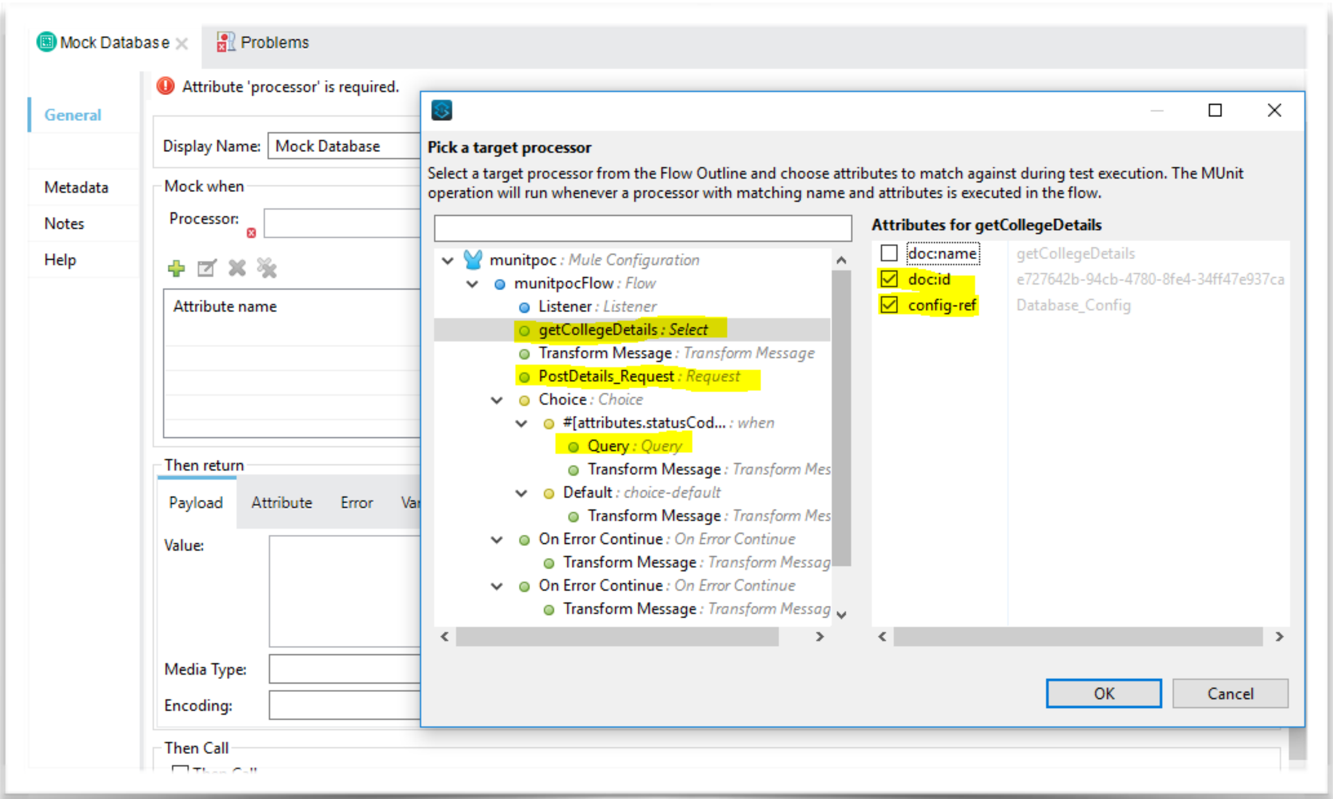
Display Name: Set Event

Set Event
Define the new event that will be output by this operation

[✓ Start with an empty event]
Clone the incoming event and override values

Value:
```
#[{
queryParams : {
            branch : "electronics"
            }
}]
```
Media Type:
Encoding:

Mocking Connectors:
Mule 4.2.2 version made Munits to write very easy .Drag and drop Mock When connector. Click on pick processor (this option is not available in version 4.1.x , still you can maually write doc:name and config-ref for identification of processor). Pick your processor which you want to mock. Then select the values like config-ref,doc:name or whatever you want so that the processor is identified uniquely.



Attribute 'processor' is required.

Display Name: Mock Database

Mock when
Processor:
Attribute name

Then return
Value:
Media Type:
Encoding:
Then Call

Pick a target processor

Select a target processor from the Flow Outline and choose attributes to match against during test execution. The MUnit operation will run whenever a processor with matching name and attributes is executed in the flow.

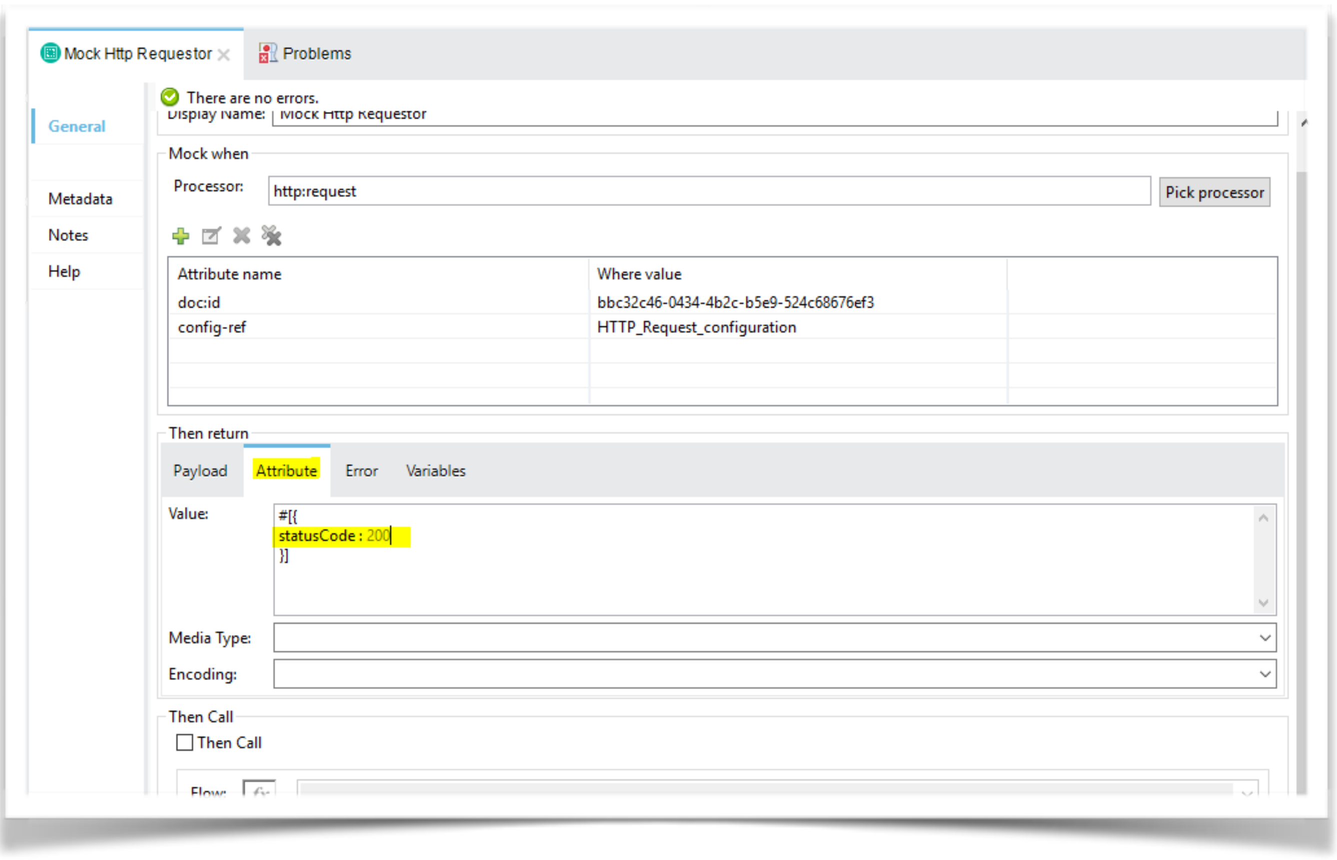
```
munitpoc : Mule Configuration
  munitpocFlow : Flow
    Listener : Listener
    getCollegeDetails : Select
    Transform Message : Transform Message
    PostDetails_Request : Request
    Choice : Choice
      #[attributes.statusCod... : when
        Query : Query
        Transform Message : Transform Mes
      Default : choice-default
        Transform Message : Transform Mes
    On Error Continue : On Error Continue
      Transform Message : Transform Messag
    On Error Continue : On Error Continue
      Transform Message : Transform Messag
```

Attributes for getCollegeDetails

☐	doc:name	getCollegeDetails
☑	doc:id	e727642b-94cb-4780-8fe4-34ff47e937ca
☑	config-ref	Database_Config

Same way if you want to mock attributes. Set as below:

Mock Attributes



Mock Http Requestor | Problems

General | Metadata | Notes | Help

There are no errors.

Display Name: Mock Http Requestor

Mock when

Processor: http:request

Attribute name	Where value
doc:id	bbc32c46-0434-4b2c-b5e9-524c68676ef3
config-ref	HTTP_Request_configuration

Then return

Payload | Attribute | Error | Variables

Value:

```
#[{
statusCode : 200
}]
```

Media Type:

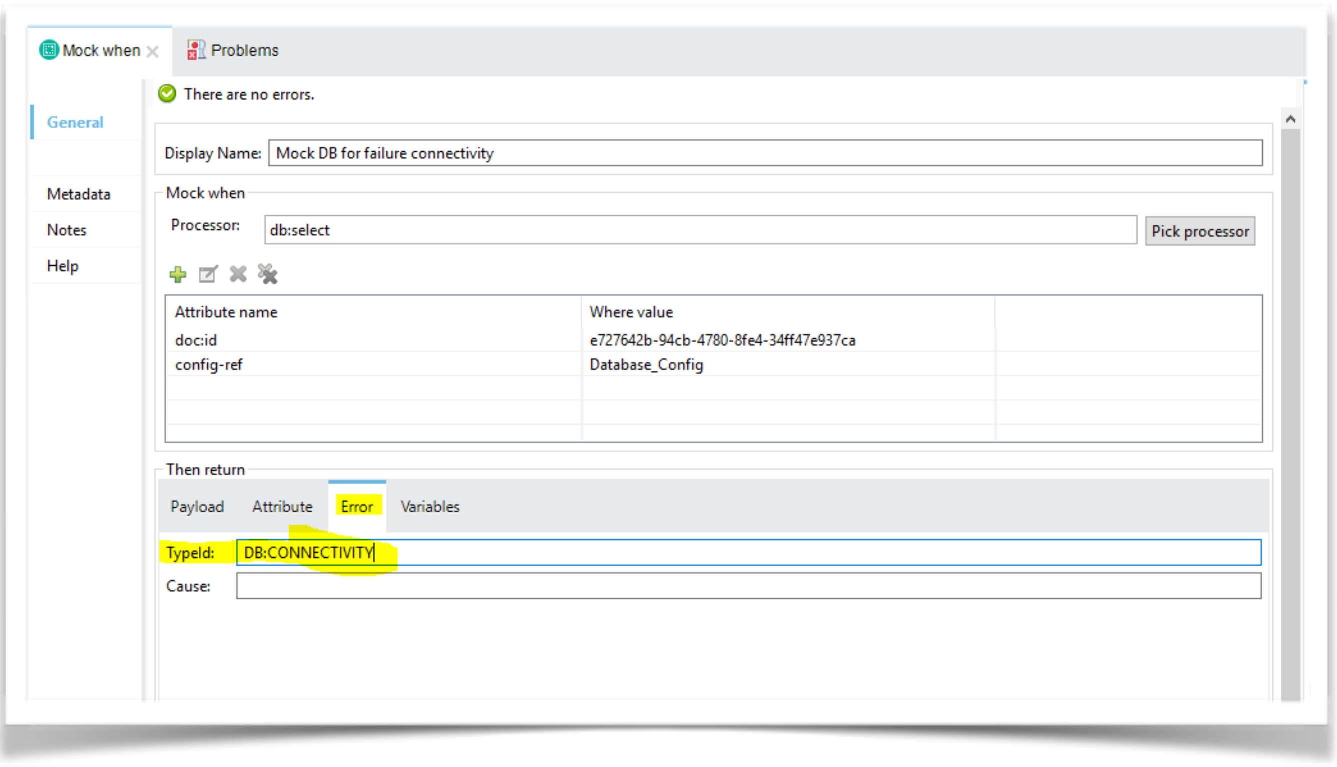
Encoding:

Then Call

Then Call

Flow:

Similarly for Error Scenarios. Mock errors as below:



Mock when | Problems

General | Metadata | Notes | Help

There are no errors.

Display Name: Mock DB for failure connectivity

Mock when

Processor: db:select

Attribute name	Where value
doc:id	e727642b-94cb-4780-8fe4-34ff47e937ca
config-ref	Database_Config

Then return

Payload | Attribute | Error | Variables

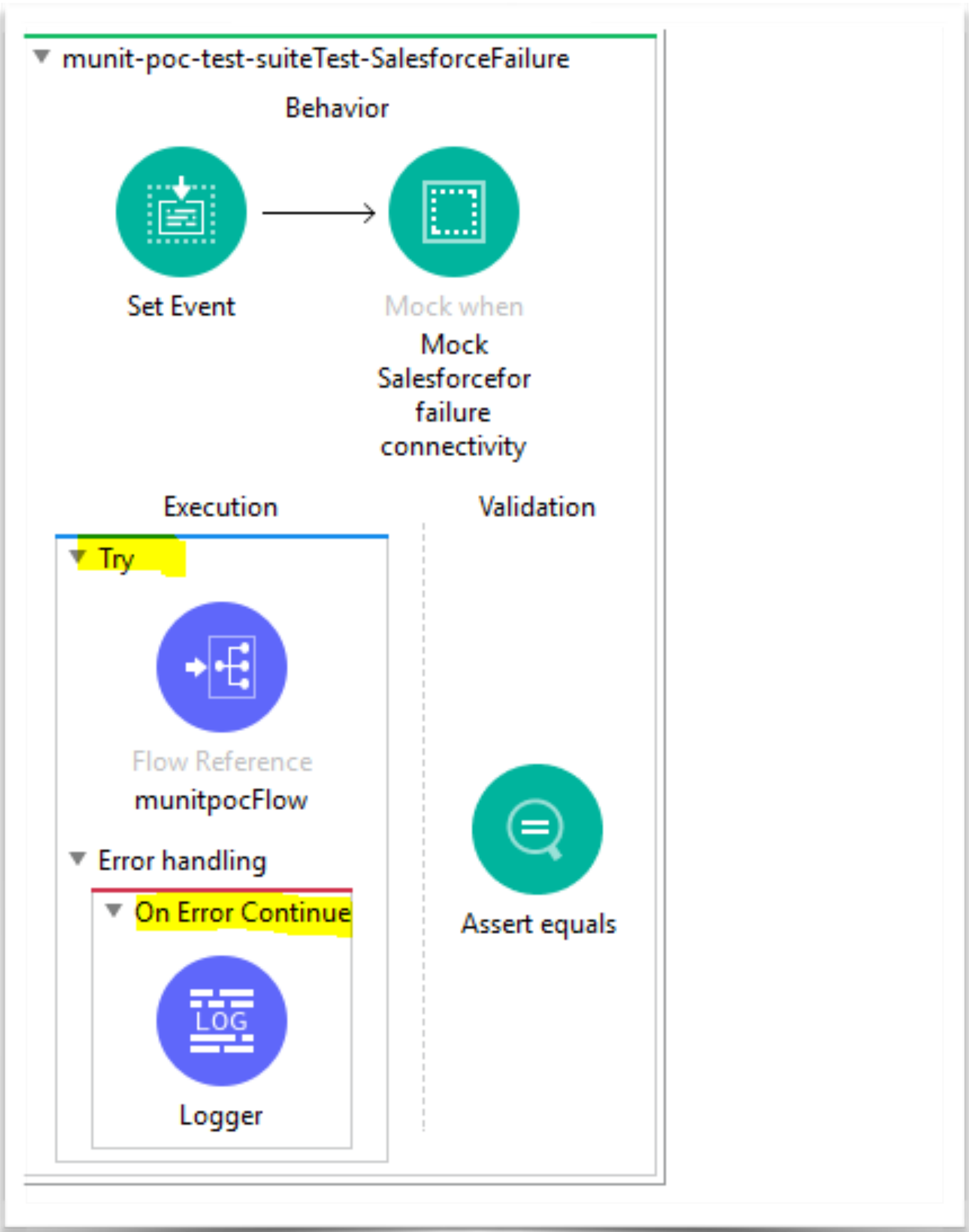
TypeId: DB:CONNECTIVITY

Cause:

You can have full code at the end of this article.
Now finally Use Assert cases to test if the value coming from flow is as expected.
That's it . Your Munit's are done!
Pretty simple!!

Things to Take care:

1) Whenever you are using Error Propagate Please make sure you are wrapping the flow-ref in execution within Try block and use error-continue . Because we are aware that when we use error-propagate , it throws an error to calling person(in this case flow-ref). see below image:



2) See the connectors properly and check what is the data that is required to mock.
3) Give proper Asserts to run Test case with success status.
4) You can ignore any flows or files using <ignoreFlows> <ignoreFiles>(refer to pom.xml). But keep one thing in mind. **The Coverage Percentage that shows in HTML page is including the excluded files and flows.** So don't worry when you see that you have ignored some files/flows and still see that % coverage is not growing. The excluded flows and actual percentage can be seen in console when you run the application. Once the % of coverage is covered as you have mentioned in pom, the application starts deploying. If not, the build will fail.
5) Final thing, to run application including test cases, Go To Windows -> preferences -> maven and remove "**-DskipMunitTests**" . So that Munits are not Skipped.
6) Last But not least, **NEVER EVER USE** doc:id as your identification factor for pick processor. doc:id changes whenever you cut/copy/paste. Also **DO NOT CHANGE** your doc:name / config-ref names after you have written MUnits. If at all there is a need in changing the names, make sure that you change them in MUnits as well

Deployments

We have various kinds of deployments options . Let's go over little description on each deployment . We will cover CloudHub and On-prem deployments in this documentation .

★ **CloudHub Deployments :**

CloudHub is an Integration Platform as a Service (iPaaS) which allows you to deploy your applications through Runtime Manager. The total Infrastructure is managed and controlled by MuleSoft cloudhub.

★ **Hybrid / On-Prem Deployments :**

These deployments can me managed to deploy your applications through Runtime manager to your Customer hosted infrastructure . Here the total infrastructure should be managed by customer hosted environment .

★ **Anypoint Runtime Fabric Deployments :**

Runtime Fabric is a container service. It runs on customer hosted Infrastructure like AWS, Azure or any other VM's(Virtual Machines). Anypoint RTF has built in scripts for AWS and Azure infra related deployments to reduce the effort .

★ **Anypoint Platform Private Cloud Edition :**

This is totally managed and controlled by customer. Usually if your customer has strict regulatory or compliance requirements that limits us to use cloud solutions . Then we go for this option

Let's see how we can deploy our application via CloudHub and On-prem.

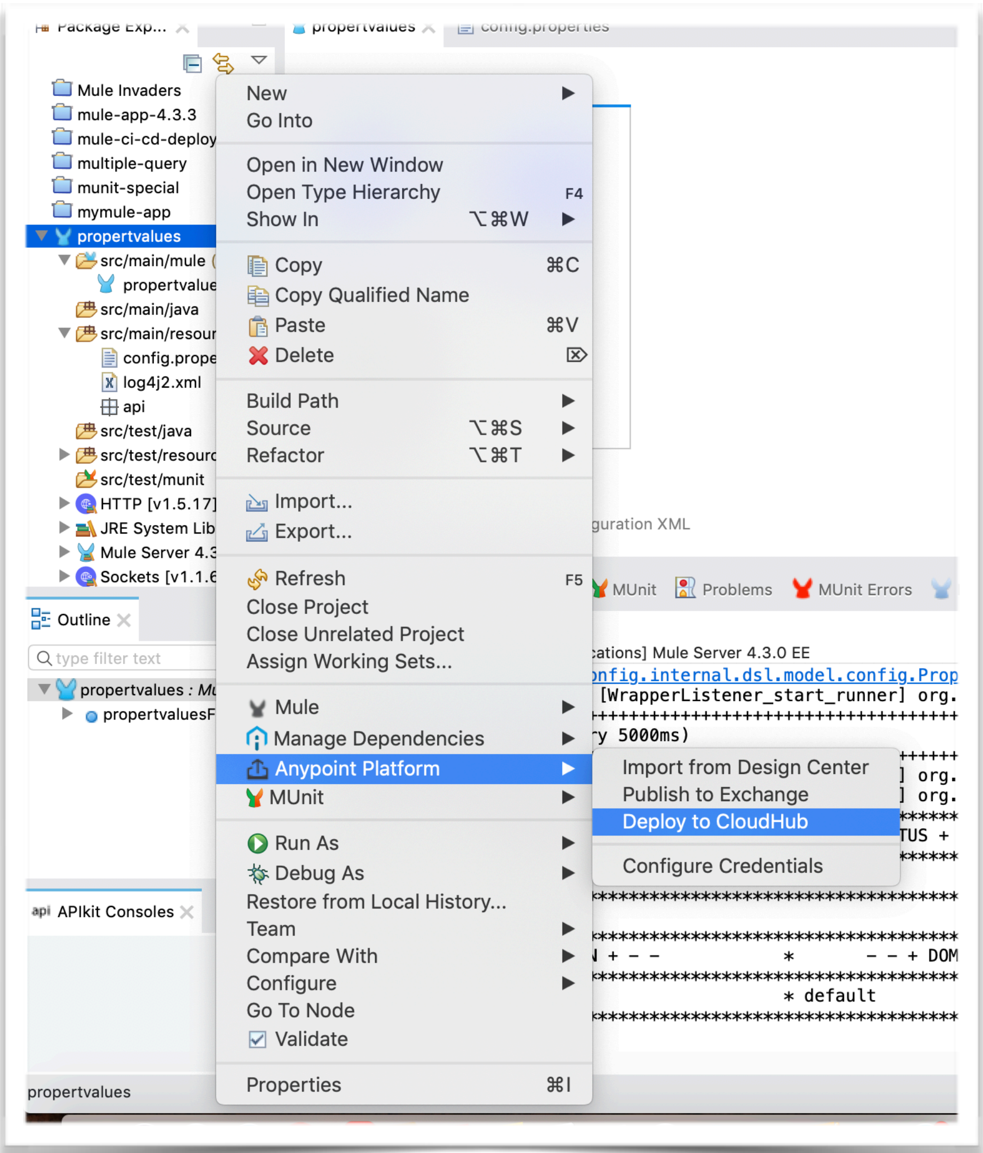
You can use CI/CD process to deploy your applications .

Deploying via CloudHub :

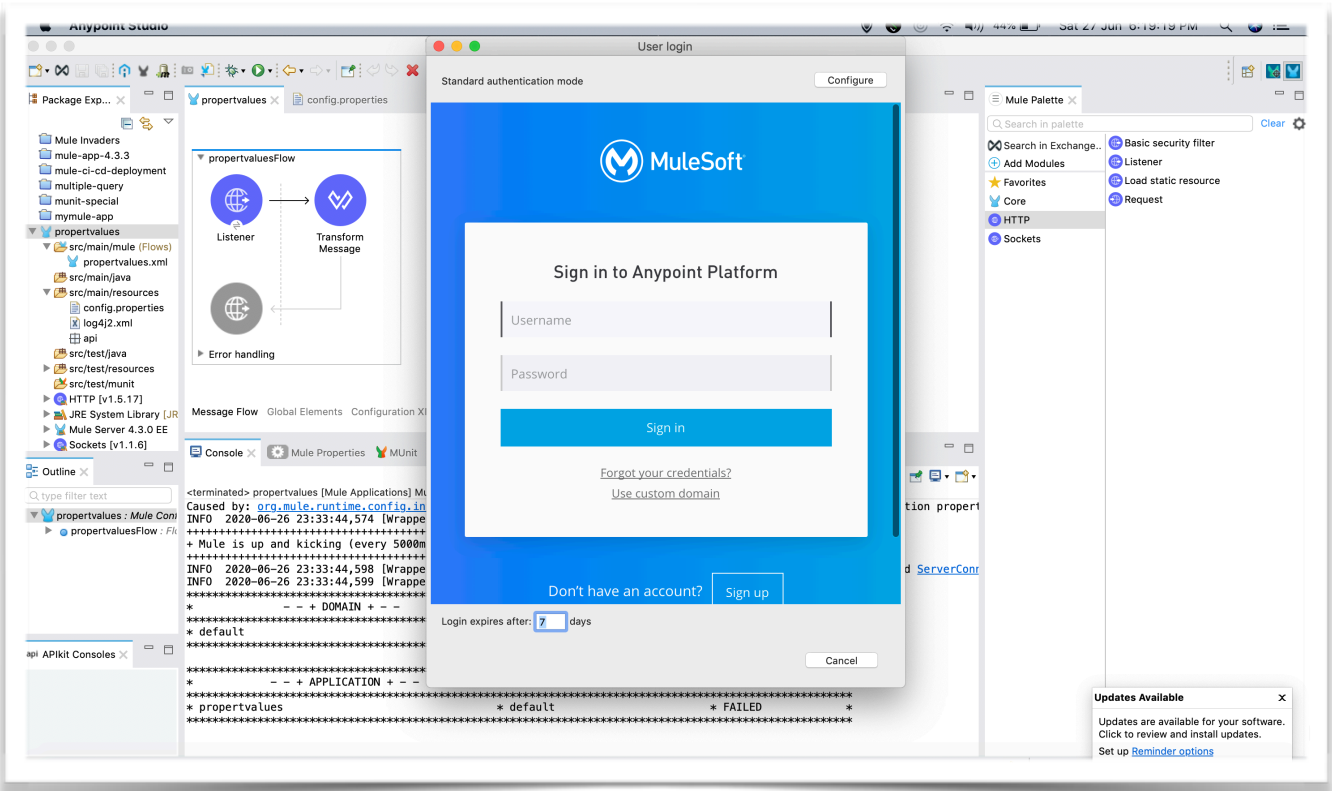
But for manual deployment , we have 2 ways. Either deploy through Anypoint Studio directly   or login to Anypoint platform and deploy through Runtime Manager.

Deploying from Anypoint Studio :

Step 1 : Once your coding is done , right-click on project —> Anypoint Platform —> Deploy to CloudHub. You have to login to Anypoint platform with your credentials in Studio.



```
cations] Mule Server 4.3.0 EE
onfig.internal.dsl.model.config.Prop
 [WrapperListener_start_runner] org.
+++++++++++++++++++++++++++++++++++++
y 5000ms)
++++++
] org.
] org.
******
TUS +
******
*************************************
*************************************
N + - -          *         - - + DOM
*************************************
              * default
*************************************
```

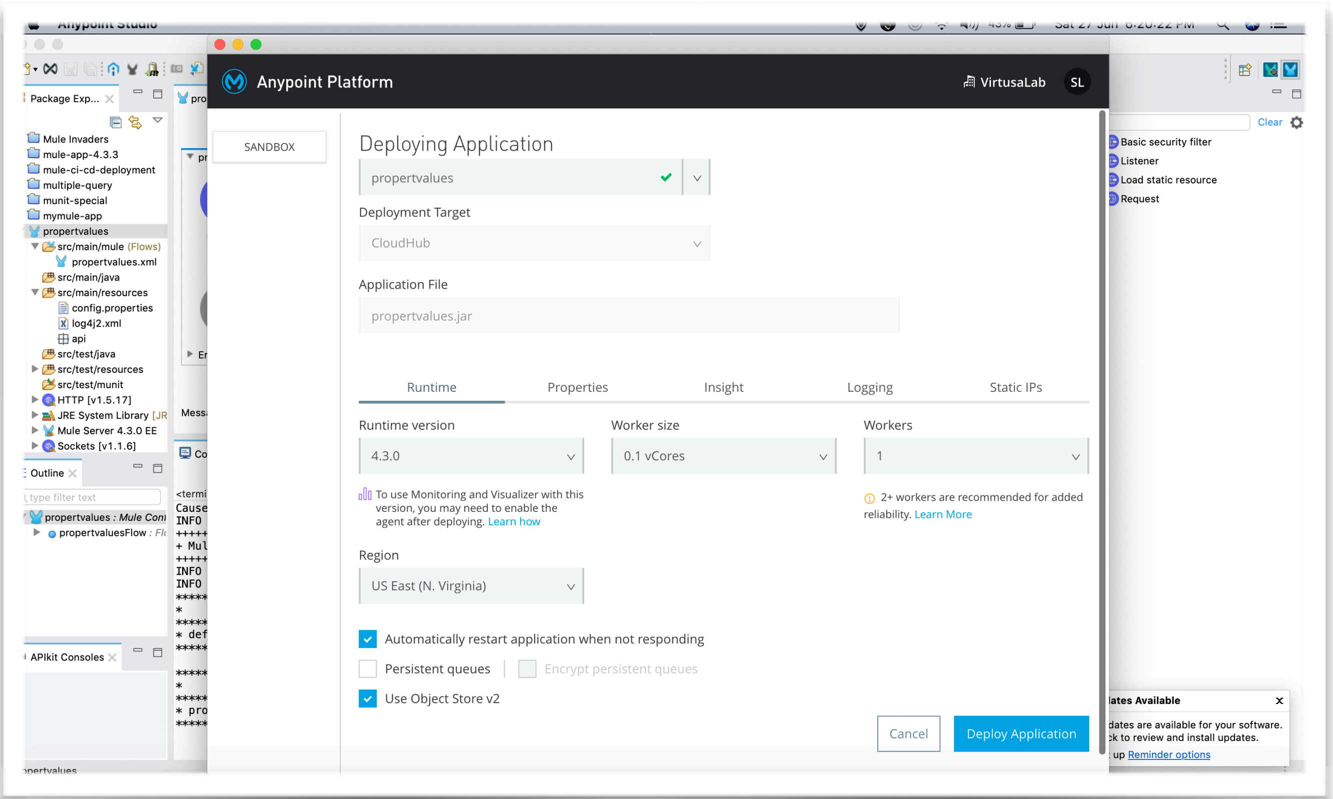


```
<terminated> propertvalues [Mule Applications] Mu
Caused by: org.mule.runtime.config.in
INFO  2020-06-26 23:33:44,574 [Wrappe
+++++++++++++++++++++++++++++++++++++
+ Mule is up and kicking (every 5000m
+++++++++++++++++++++++++++++++++++++
INFO  2020-06-26 23:33:44,598 [Wrappe
INFO  2020-06-26 23:33:44,599 [Wrappe
*************************************
*            - - + DOMAIN + - -
*************************************
* default
*************************************

*************************************
*          - - + APPLICATION + - -
*******************************************************************************
* propertvalues                       * default                   * FAILED      *
*******************************************************************************
```

```
tion propert
d ServerConf
```

Step 2 : Now After login from studio. Runtime Manager's Deploy Application screen will be prompted. By default the application name is set to whatever name you have given in your project . You can change it if you want.



Anypoint Platform

SANDBOX

Deploying Application

propertyvalues

Deployment Target: CloudHub

Application File: propertyvalues.jar

Runtime | Properties | Insight | Logging | Static IPs

Runtime version: 4.3.0

Worker size: 0.1 vCores

Workers: 1

To use Monitoring and Visualizer with this version, you may need to enable the agent after deploying. Learn how

2+ workers are recommended for added reliability. Learn More

Region: US East (N. Virginia)

- [✓] Automatically restart application when not responding
- [ ] Persistent queues
- [ ] Encrypt persistent queues
- [✓] Use Object Store v2

Cancel | Deploy Application

Step 3 : Click on "Deploy Application " . That's it your application will be build and deploy at a time!

Deploying from Anypoint Platform :

To perform this step, first you need to run your application on your local, so that the jar file is created under "target" folder of your project.

✓ Now Goto Anypoint Platform —> Runtime Manager—>Choose your environment

✓ Click on Deploy Application —> Give the application name and Deployment Target as "CloudHub" and click on Choose File —>Upload file —> Select the jar file which is generated in target folder —> Choose your Runtime option and then click on Deploy Application . It's done!

Note : The worker size and workers can be increased based on your application size and performance.

Note : Default http and https ports for cloudhub deployments are 8081 and 8082 respectively . So its always better to define ${http.port} or ${https.port} and leave the host name as is to default .

Deploying via On-prem :

If you opt for On-prem deployment , then you need to setup Mule Server on your environment . For this , you have to download Mule Standalone Server.
Download from : https://www.mulesoft.com/lp/dl/mule-esb-enterprise

As we saw two ways in deploying the applications via cloudhub , we have 2 ways here as well.

Deploying Applications directly via "apps" folder :

Step 1 : Once you download the Mule Standalone server, extract it . Now open command prompt (if your system is running on windows) or terminal (if using MacOS). Go to bin folder path of Mule Standalone Server and run **mule.bat**
You can now see that your Mule Server is up and running , and is ready to deploy your applications

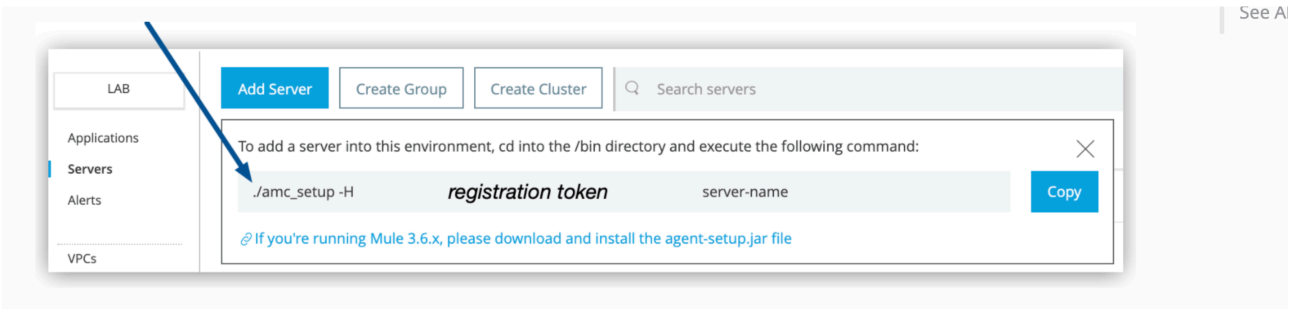
Step 2:

- ✓ Now it's very easy to deploy your application on on-prem. Just copy paste the jar file that is generated in your "target" folder into "apps" folder of Mule Standalone Server .
- ✓ This will automatically start deploying your application . You can check that in command prompt or terminal.
- ✓ Once it's deployed , you can see an anchor text file created with the project name in "apps" folder.
- ✓ If at all you want to delete any application , simply delete this text file in apps folder.

Deploying Applications directly via Runtime Manager:

Step 1: Login to Anypoint platform . Goto Runtime Manager —> Choose environment —> click on Servers.

Step 2 : Now Click on add server . You can see a command where you can copy it and run on our customer hosted environment.



To add a server into this environment, cd into the /bin directory and execute the following command:

```
./amc_setup -H registration token server-name
```

If you're running Mule 3.6.x, please download and install the agent-setup.jar file

Step 3: Goto command prompt or terminal , goto bin folder path of Mule Standalone server and paste the command that you copied in Step 2 .
You can give a proper server name which you would like to give. Remove . / and run the command

Step 4:
You can see that "Mule Agent configured successfully" is prompted on screen
Basically what we have done is , we are creating a connection between our on-prem server to Runtime manager. So basically we need a Mule agent to mediate.

Step 5:
Simple, If you want to deploy the application ,
Go to Runtime Manager —> Choose environment —>Deploy new application —> Select the Deployment target shown in drop down list with your server name —> Choose file from local (jar file) and Click on Deploy Application .

Step 6 :
That's it your application is successfully deployed on on-prem. You can also check in "apps" folder that anchor text file is created. If at all you want to delete the app, we can delete that particular anchor text file.

Note : There's no Default http and https ports for on-prem deployments . You can define your own ports which is valid for your infrastructure .

Also if you want multiple servers. (Like workers in cloudhub deployments)

Have a copy of Mule Standalone servers with different names and follow Step 2,3,4 to create multiple servers.

This is how you deploy your applications both in CloudHub and On-prem

Managing Applications by applying Policies:

Note : Anypoint Platform's API Manager polices can only be applied to the applications which are deployed only on CloudHub. For other mode of deployments , you need to have other ways of applying policies .

This Documentation shows you how to apply policies via Anypoint Platform's API manager.

Again , we have 2 ways to apply policies :

- **Creating Proxy API's**
- **Without Creating Proxy API's**

The main difference between two of them is either :

Using API id in Auto Discovery of your "actual" application
or
Mule itself will create a Proxy application on top of your "actual" application which has api auto discovery.

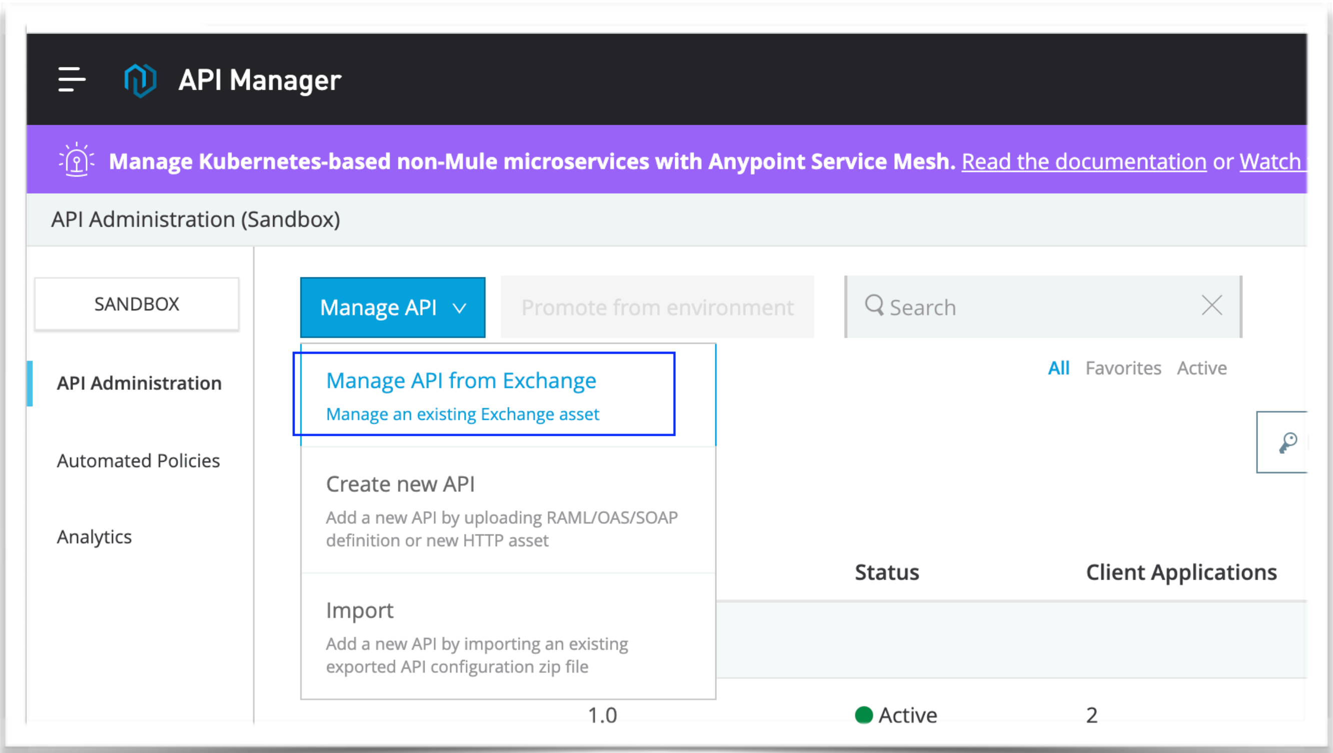
We shall see how in detail:

- **Without Creating Proxy API's :**
  This way of Deploying application to invoke policies needs 2 Things:
  - API id generated in API Manager and used in global element "Auto Discovery" of your application .
  - Need Anypoint platforms client id and secret .We can get them from "Access Management" Under Environments Tabs —> Choose the environment in which your application is getting deployed. Use them in Runtime properties while deploying application in below way
    anypoint.platform.client_id : xxxxxxxxxxxx
    anypoint.platform.client_secret : xxxxxxxxxxxx

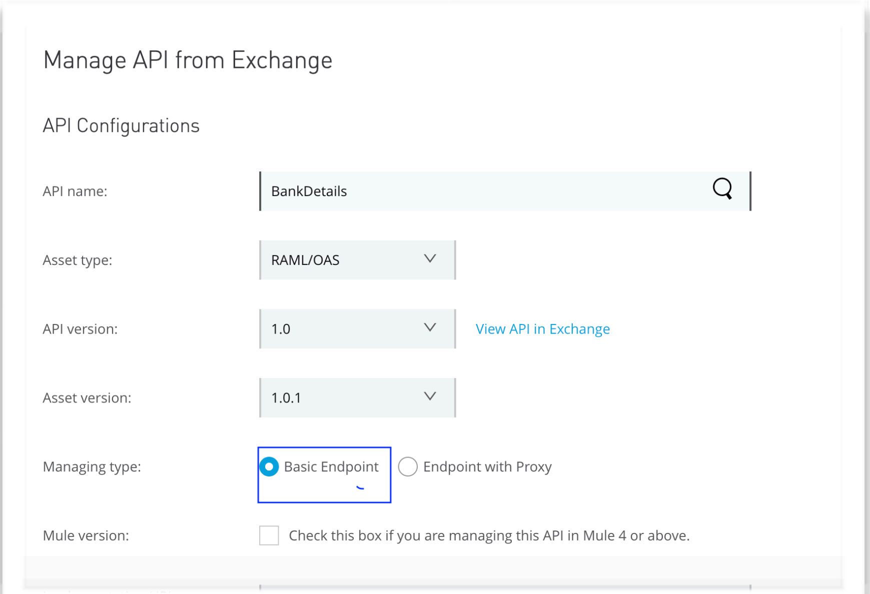
With help of these 2 things, we can see in API Manager , the status of API will turn from "Unregistered" to "Active"

Step 1 : Design a RAML , Publish to exchange. After that import the RAML to anypoint Studio and generate the flows.

Step 2 : Go to API Manager , Click on Manage API and choose "Manage API from Exchange"

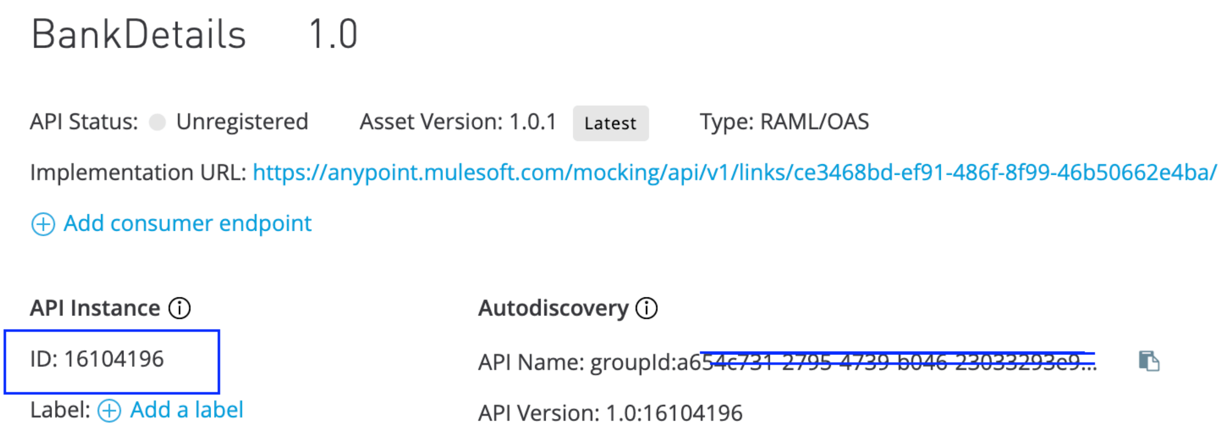

API Manager
Manage Kubernetes-based non-Mule microservices with Anypoint Service Mesh. Read the documentation or Watch
API Administration (Sandbox)
SANDBOX
API Administration
Automated Policies
Analytics
Manage API
Promote from environment
Search
Manage API from Exchange
Manage an existing Exchange asset
Create new API
Add a new API by uploading RAML/OAS/SOAP definition or new HTTP asset
Import
Add a new API by importing an existing exported API configuration zip file
All
Favorites
Active
Status
Client Applications
1.0
Active
2

Step 3 : Select your API , And select the Managing type as "Basic Endpoint" . Don't forget to check the box "Mule version: Check this box if you are managing this API in Mule 4 or above."
Click on Save ,


Manage API from Exchange

API Configurations

API name:	BankDetails
Asset type:	RAML/OAS
API version:	1.0
Asset version:	1.0.1
Managing type:	[✓ Basic Endpoint] Endpoint with Proxy
Mule version:	Check this box if you are managing this API in Mule 4 or above.



BankDetails 1.0

API Status: Unregistered Asset Version: 1.0.1 Latest Type: RAML/OAS

Implementation URL: https://anypoint.mulesoft.com/mocking/api/v1/links/ce3468bd-ef91-486f-8f99-46b50662e4ba/

Add consumer endpoint

API Instance

ID: 16104196

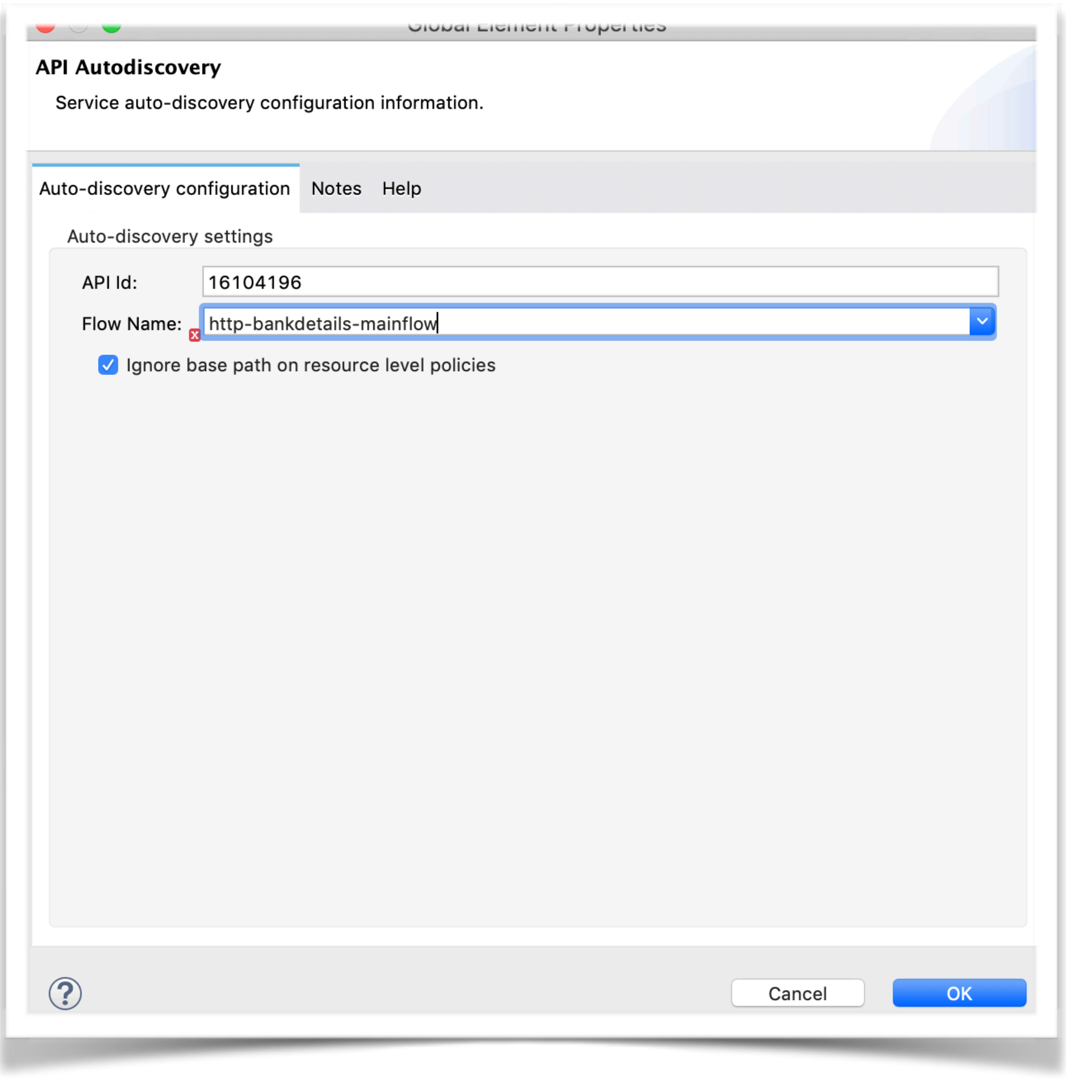
Label: Add a label

Autodiscovery

API Name: groupId:a65~~4c731-2795-4739-b046-23033293e9...~~

API Version: 1.0:16104196

Step 4 : Go to the Application in Anypoint Studio, Create a Global Element Auto Discovery and Add API id and choose the flow name (usually Main flow where the request is listened)



Global Element Properties

API Autodiscovery

Service auto-discovery configuration information.

Auto-discovery configuration | Notes | Help

Auto-discovery settings

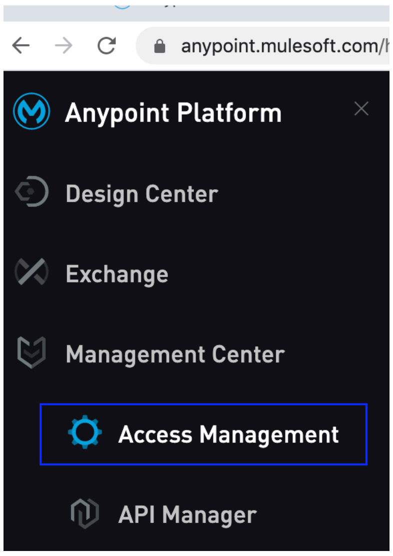
API Id: 16104196

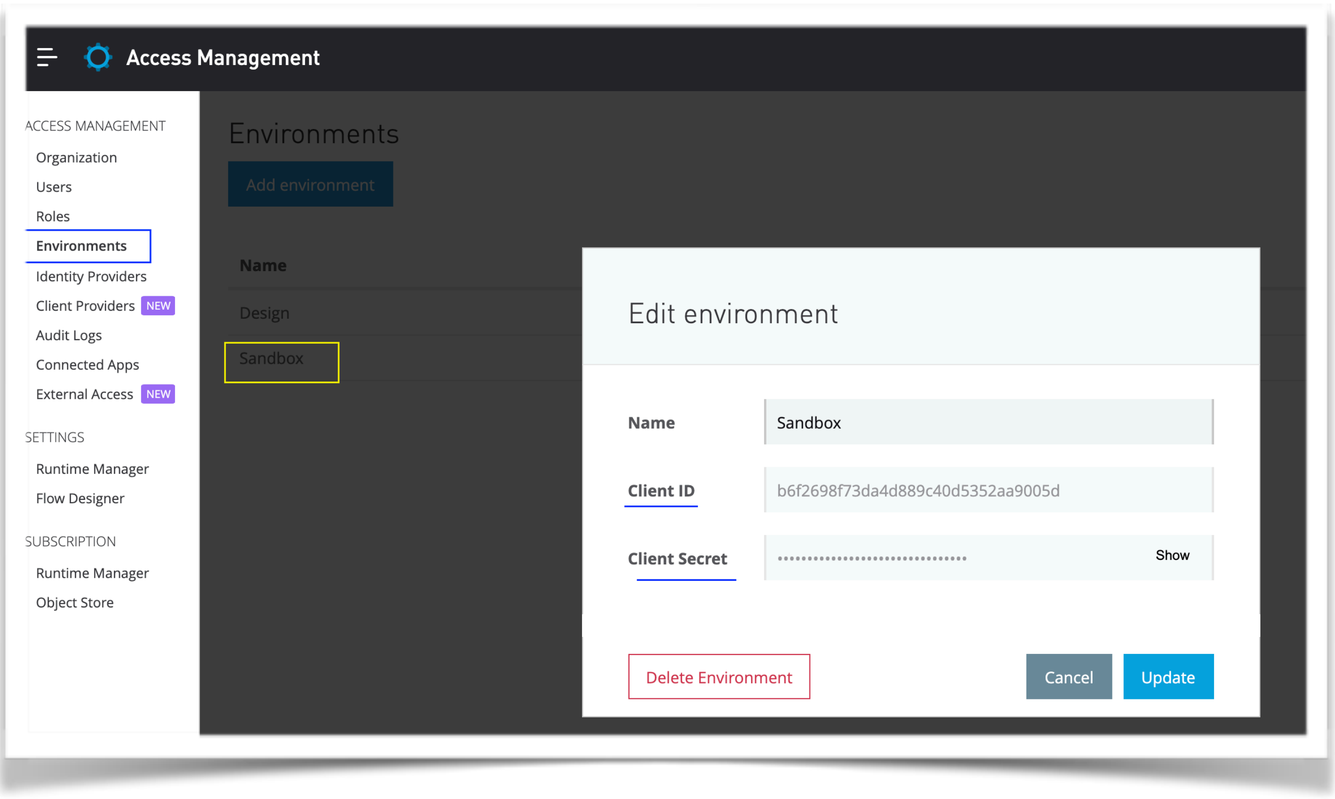
Flow Name: http-bankdetails-mainflow

☑ Ignore base path on resource level policies

Cancel | OK

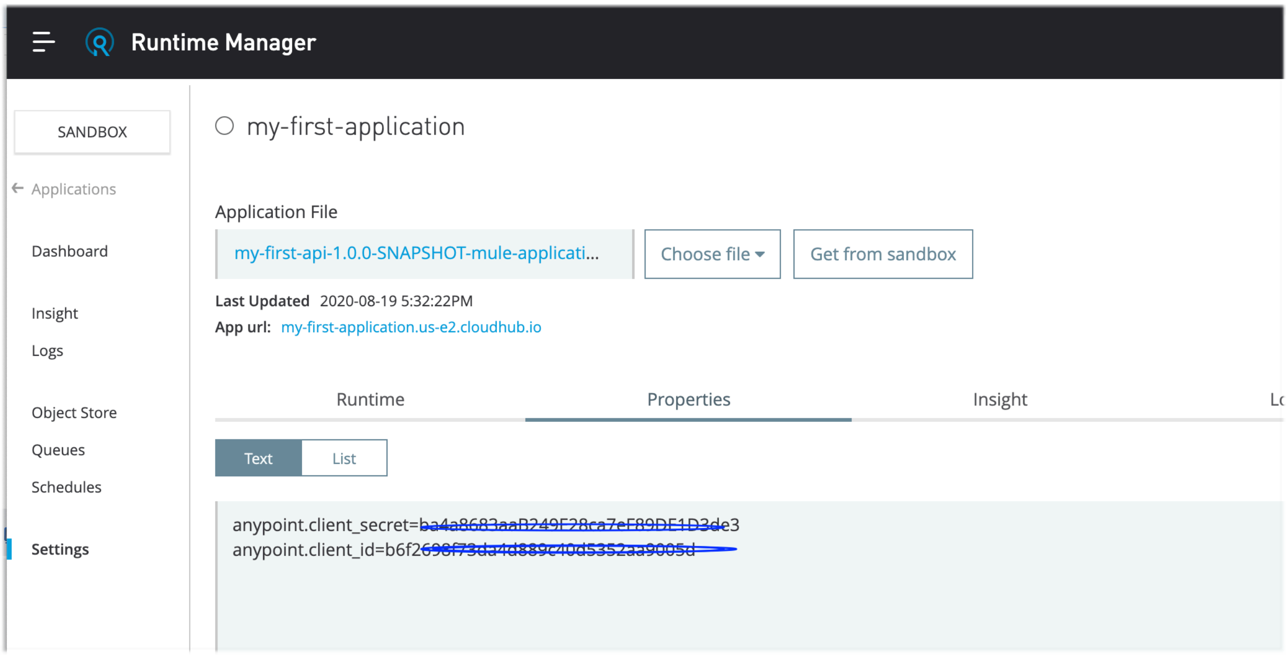
Step 5: Go to Access Management.(Only Owner of the account can have access and can view this option. If you are not seeing this, that means you need to request the owner of Anypoint platform account) . Choose environments Tab and select your env. Get your client id and secret





Name: Sandbox
Client ID: b6f2698f73da4d889c40d5352aa9005d
Client Secret: ••••••••••••••••••••••••••••••••

<u>Step 6 :</u> Go to Runtime Manager , Add Runtime properties like below and deploy.



```
anypoint.client_secret=[redacted]
anypoint.client_id=b6f2[redacted]
```

Now your API in API manager status will turn to **Active** and you can start applying policies. By going to Policies —> Apply New Policy.

No Need to Restart the application once policy is applied. It will reflect within seconds automatically . Even if you add one or more policies at later point of time, no restart of application is required.

Creating Proxy API's:

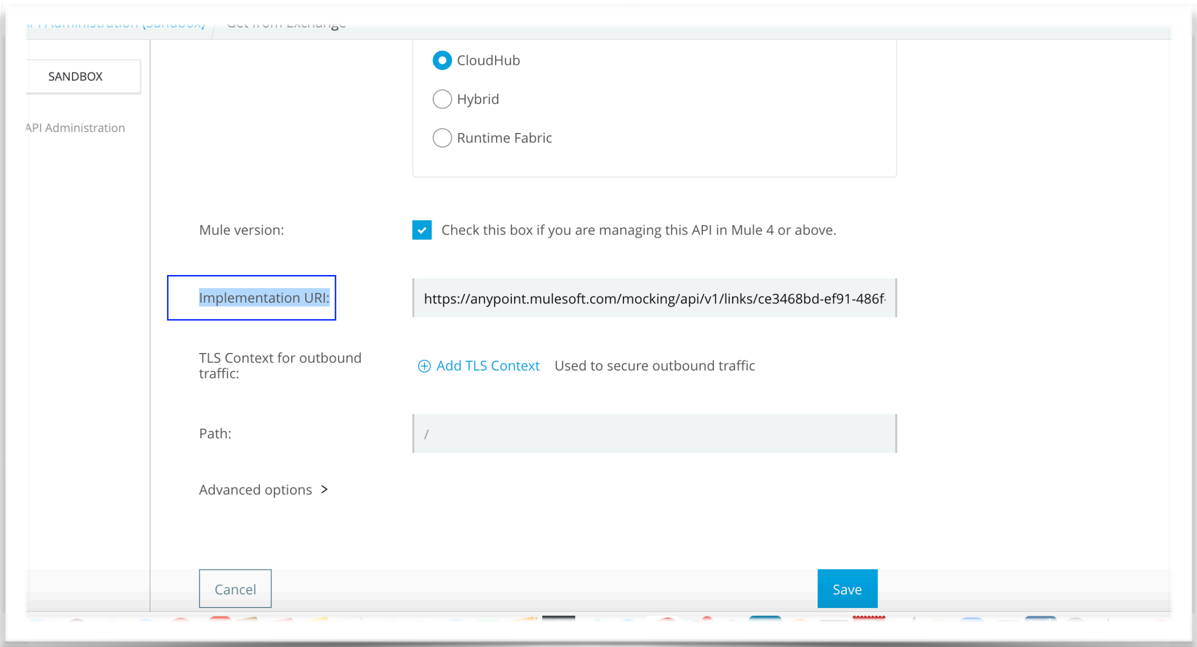
In above method, we have seen lot of manual process required like getting API id , clientid,secret , placing in Application and in Runtime properties. To avoid all these, we can go for 2nd approach of create a Proxy application .
This Proxy application is nothing but a kind of Wrapper to your main Application . So now for each application we will have 2 apps in Runtime Manager.
1) Your Actual Application
2) Your proxy Application that is deployed which has api id, client id , secret everything within it. No manual effort required to deploy. It's deployed from API manager.

<u>Step 1 :</u> Design a RAML , Publish to exchange. After that import the RAML to anypoint Studio and generate the flows.

<u>Step 2 :</u> Go to API Manager , Click on Manage API and choose "Manage API from Exchange"

Step 3 : Select your API , And select the Managing type as "Endpoint with Proxy" . Don't forget to check the box "Mule version: Check this box if you are managing this API in Mule 4 or above." Click on Save ,

Step 4 : Give your Implementation URL of your actual application that is deployed.
Make sure you give your path till /api as all RAML generated flows has a common base path as /api
Eg : http://myfirstapp.cloudhub.io/api
Once done, click on save


CloudHub [✓]
Hybrid
Runtime Fabric

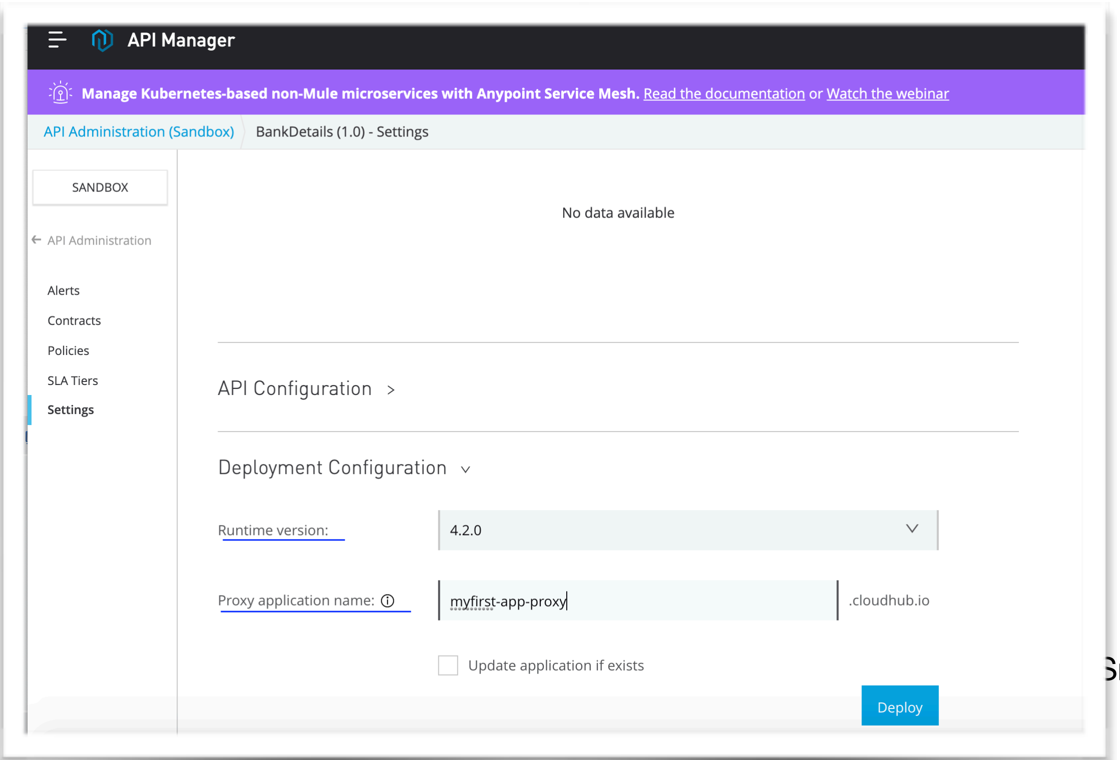
Mule version: [✓] Check this box if you are managing this API in Mule 4 or above.

Implementation URI: https://anypoint.mulesoft.com/mocking/api/v1/links/ce3468bd-ef91-486f

TLS Context for outbound traffic: Add TLS Context Used to secure outbound traffic

Path: /

Step 5 : Now on the screen, you can see Deployment Configuration . Choose runtime and name of proxy app you want to give and click on deploy. The Application will be deployed in Runtime Manager.


API Manager

Manage Kubernetes-based non-Mule microservices with Anypoint Service Mesh. Read the documentation or Watch the webinar

API Administration (Sandbox) BankDetails (1.0) - Settings

No data available

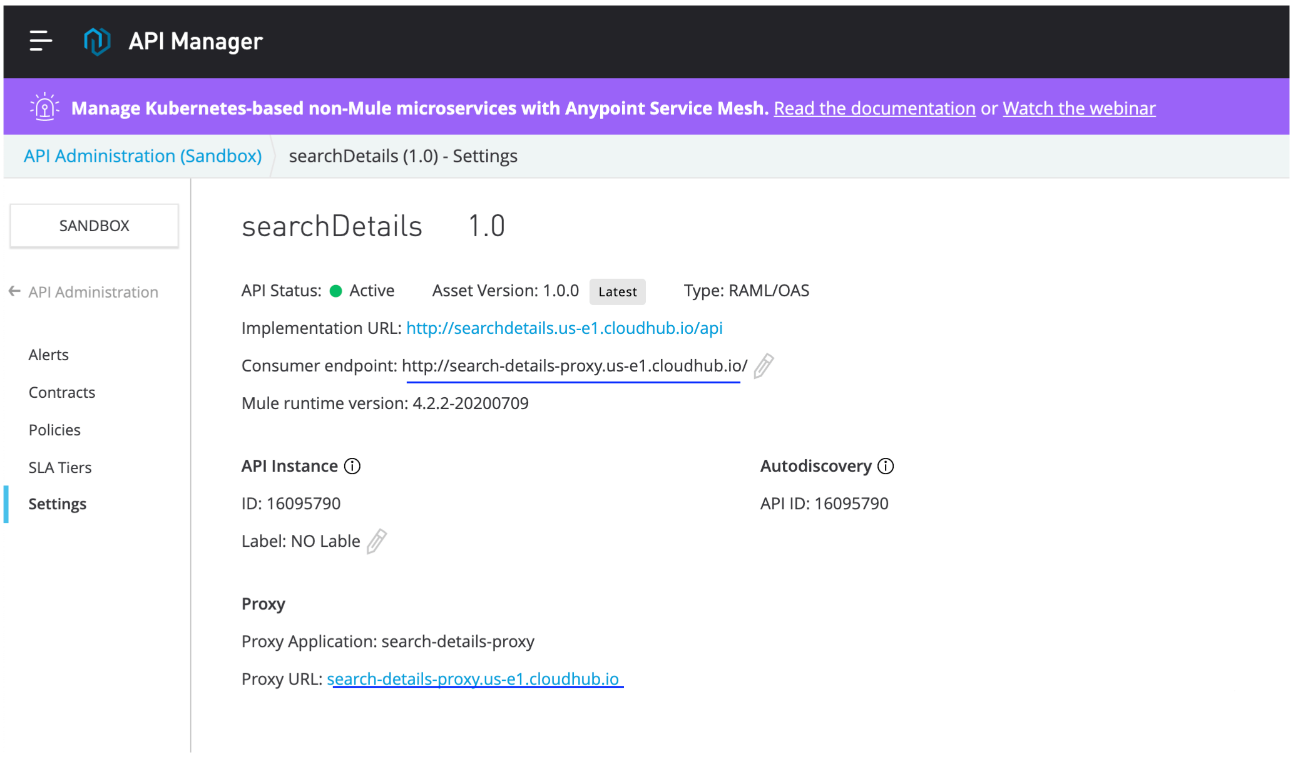
Deployment Configuration

Runtime version: 4.2.0

Proxy application name: myfirst-app-proxy .cloudhub.io

Update application if exists

Step 6 : Now edit the Consumer end point in API Manager with your proxy endpoint. So that all the requests will hit proxy application . Basically your implementation url is your actual endpoint and Consumer endpoint is Proxy endpoint. So client hits proxy endpoint which in-turn is routed to implementation url


API Manager
Manage Kubernetes-based non-Mule microservices with Anypoint Service Mesh. Read the documentation or Watch the webinar
API Administration (Sandbox)
searchDetails (1.0) - Settings
SANDBOX
API Administration
Alerts
Contracts
Policies
SLA Tiers
Settings

searchDetails 1.0

API Status: ● Active Asset Version: 1.0.0 Latest Type: RAML/OAS

Implementation URL: http://searchdetails.us-e1.cloudhub.io/api

Consumer endpoint: http://search-details-proxy.us-e1.cloudhub.io/

Mule runtime version: 4.2.2-20200709

API Instance

ID: 16095790

Label: NO Lable

Autodiscovery

API ID: 16095790

Proxy

Proxy Application: search-details-proxy

Proxy URL: search-details-proxy.us-e1.cloudhub.io

You can see client id and secret values are picked automatically and placed in Runtime properties. As well as you can download the jar of proxy app and import in anypoint studio where you can see api Id is automatically configured and the Http request will route to your main APP.
So you have to provide endpoint of Proxy app to consumers/clients so that they don't have access to your main App. The policies are applied to your proxy app.
But thing here is, we need additional vCore and worker to deploy our proxy app.
So according to your requirements you can choose which method you want to follow to apply policies for your cloudhub Apps.

Pros and Cons :

Without Proxy Application :

- You will save a vCore and worker .
- As a developer You need to know client id / Secret unless the deployment is managed by CI/CD process.
- Manually you need to create global element to invoke Auto discovery
- You will expose your actual endpoint to client/Consumer.
- Use case : If your apis are used within organisation , within Intranet , you can choose this option where we can share the endpoints directly. This saves vCores

With Proxy Application :

- You will need additional vCore and worker for each proxy app you deploy.
- As a developer You need not know client id as it is managed internally.
- No need of manual process of using api id or using client id / secret
- You will expose your proxy endpoint over actual endpoint to client/Consumer.
- Use case : If your apis are used outside of your organisation , you can choose this option where we can share the endpoints of proxy app instead of actual endpoint. More secured

Applying Policies using API MANAGER(for only Cloud-hub apps):

Simple :
After the Deployment to cloudhub :
Apply policies to the applications in a simpler way.
If you have Proxy App , then apply policies to Proxy application , else apply for root app.
Note we apply policies for HTTP Services ONLY.

Now after you see the status as ACTIVE in API Manager, Now we are ready to apply policies.
Go to Policies Tab of that particular Application —> click on Add New Policy (you can apply more than one) —> Choose your policy and click Apply.

That's it, no need to restart your application to get policies appended. It will be appended in few seconds.

Note : If your API is not showing ACTIVE in API Manager, that means , either the api ID is mismatched that you have given in your code and the value in API Manager, or The Anypoint platform credentials are wrong. When Status is "Un Registered " That means , your policies are not appended. But still application works without having Policies appended.

For On-prem Apps, you have to have a different kind of mechanism within code to apply policies , as API Manager policies are applicable only for Cloudhub deployed Apps.