# 1st Udemy Course :[MuleSoft 4 Fundamentals - Beginner to Expert Level](https://deloittedevelopment.udemy.com/course/mulesoft-beginners-to-expert-mule-42/)

**Anypoint platform uname: surya\_practice**

**Pwd: Practice@123**

**Trial Expiration Date: 2023-10-10**

-

Setup Requirements:

1)Download Anypoint Studio

2)Download Postman/Advance Rest Client to Test the Webservices

3)Should have Anypoint Platform Account

HTTP-Listener: Used as listener for webservices call.

-HTTP Listener listens for HTTP requests and triggers the associated flow when it receives them.

-In HTTP-listener Config,Base-Path if set then that basepath should be use as it is in API Request URL after host n port name,if not then ignore basepath from below syntax

Protocol://host:port/basepath/path?queryparams(here basepath ,path n queryparams can be optional)

For eg:-

<http://localhost:8081/surya-sys-learn-udemy/demo?name=suryakant&age=29&company=Deloitte>

Note:

1. in http Listener config, We can specify basepath as /surya-sys-learn-udemy or surya-sys-learn-udemy both are same considered so in the API URL it will automatically adds /surya-sys-learn-udemy if we haven’t specified like /surya-sys-learn-udemy
2. in http Listener config, We can specify basepath as /demo or domo both are same considered so in the API URL it will automatically adds /demo if we haven’t specified like /demo n if we hit below endpoint then http listener will listen to that call

<http://localhost:8081/surya-sys-learn-udemy/demo?name=suryakant&age=29&company=Deloitte>

-In case if we give don’t specify any path then anypoint studio will prompt attribute path required and if we provide path as / then http listener will listen to all the requests where we don’t have any path after basepath(if we specified in listener config) or after host n port(if we don’t specify basepath also in config)

For eg:

[http://localhost:8081/surya-sys-learn-udemy?name=suryakant&age=29&company=Deloitte (will listen if](http://localhost:8081/surya-sys-learn-udemy?name=suryakant&age=29&company=Deloitte%20(will%20listen%20if) Path=/ n basepath= surya-sys-learn-udemy )

<http://localhost:8081?name=suryakant&age=29&company=Deloitte>  [(will listen if](http://localhost:8081/surya-sys-learn-udemy?name=suryakant&age=29&company=Deloitte%20(will%20listen%20if) Path=/ n basepath= not specified in config)

-if we want that listener should listen to all request when we don’t have basepath specified then we can specify path= /\* it will listen to all requests

<http://localhost:8081/surya-sys-learn-udemy/demo?name=suryakant&age=29&company=Deloitte>

<http://localhost:8081?name=suryakant&age=29&company=Deloitte>

<http://localhost:8081/demo?name=suryakant&age=29&company=Deloitte>

<http://localhost:8081/surya?name=suryakant&age=29&company=Deloitte>

-if we want that listener should listen to all request when we have basepath specified then we can specify path= /\* it will listen to all requests which have specified basepath in API URL(let say basepath= surya-sys-learn-udemy)

Then it will listen to all the requests that have

<http://localhost:8081/surya-sys-learn-udemy/demo?name=suryakant&age=29&company=Deloitte>

<http://localhost:8081/surya-sys-learn-udemy/suryakant?name=suryakant&age=29&company=Deloitte>

[http://localhost:8081/surya-sys-learn-udemy?name=suryakant&age=29&company=Deloitte](http://localhost:8081/surya-sys-learn-udemy/demo?name=suryakant&age=29&company=Deloitte)

-For Http-Listener, we can specify allowed methods so that it accepts only that methods like GET,POST,PUT,PATCH etc... let say if we set Allowed Method=POST and if we are triggering GET method call from postman then it will return 405-method not allowed with body Method not allowed for endpoint: /<basepath>/<path>

Note:

1. /\* will work as expected only in case we have only one http listener with path as /\* n matching methods allowed.two or more http listener with same path i.e /\* or any other path n with same methods allowed then will always give the preferences to the last matched flow with same path ,method host port.
2. We cant use /demo\* also in path, the Wildcards can only be used at the end of the path (i.e.: /path/\*) or between / characters (.i.e.: /path/\*/anotherPath)
3. IF we call child flow with http listener config(diff path) from main flow http listener config (diff path) with flow-reference then all attributes,payload,variables can be aacessed in child flow as well on the other hand if we call child-flow http listener config (diff path) from main flow http listener config (diff path) with http-request then only payload can be aacessed inside child flow whereas old attributes will be vanished n new http-request attributes will be accessible and for variables created in main flow before http request are not accessible inside child flow , but they can be accessible once the trigger comes into the main flow n let say if we create variable inside child flow then they can be accessible only till child flow completes its execution after that they cannot be accessible.

-By Default In case of success ,Http Listener provides payload as body with status code 200

-In case of failure ,Http Listener provides error.description as body with status code 500

UriParams and QueryParams:

-URI parameter (Path Param) is basically used to identify a specific resource or resources whereas Query Parameter is used to sort/filter those resources.

Let's consider an example where you want identify the employee on the basis of employeeID, and in that case, you will be using the URI param.

**GET** /employee/{employeeID}

Take another example where you want to filter the employee on the basis of designation, and in that case, you will be using Query Parameter.

|  |  |
| --- | --- |
| Return list of car having color black | GET /cars?color=black |
| Return list of cars sorted by model | GET /cars?sortBy=model |
| Return list of drivers for car 888 | GET /cars/{car-number}/drivers GET /cars/car888/drivers |
| Return employees having employeeID 10 | GET /employees/{employeeID} GET /employees/10 |

Defining URI Parameter With RAML

URI Parameter is a variable element enclosed in curly braces {} inside relative URI of resources.

|  |
| --- |
| /employees:  /{employeeID}:  description: Return employee information on basis of EmployeeID  get:  responses:  200:  body:  application/json:  example: |  {"employeeID":"1", "employeeName":"Tom Berry","designation":"SSE"} |

Looking at above RAML, the URL to get employee details on the basis of employeeID will be **GET /employees/{employeeID}**where employees is resource and {employeeID} is URI parameter**.**

## Defining the Query Parameter With RAML

Query Parameter is basically used to filter or sort the resources. It is passed in the URL as query string in key-value form.

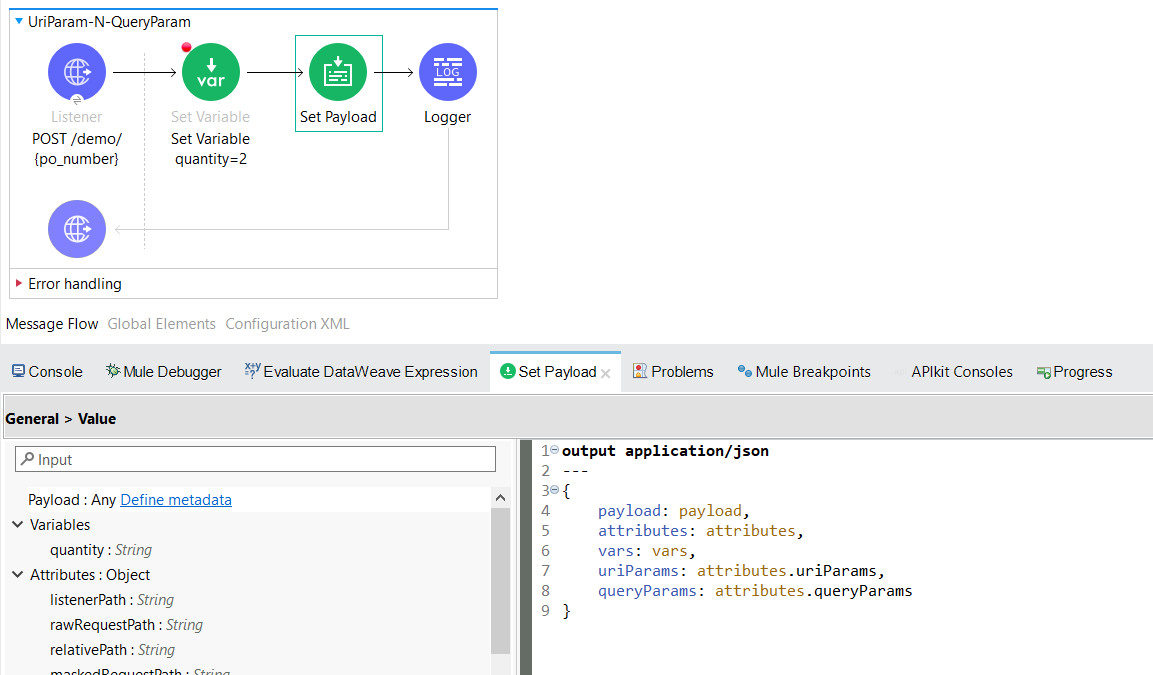
|  |
| --- |
| /employees:  get:  queryParameters:  designation:  description: Designation of employee  type: string  required: true  responses:  200:  body:  application/json:  example: |  {"employees":[  {"employeeID":"1", "employeeName":"Tom Berry","designation":"SSE"},  {"employeeID":"2", "employeeName":"JAck Jones","designation":"SSE"}  ]} |

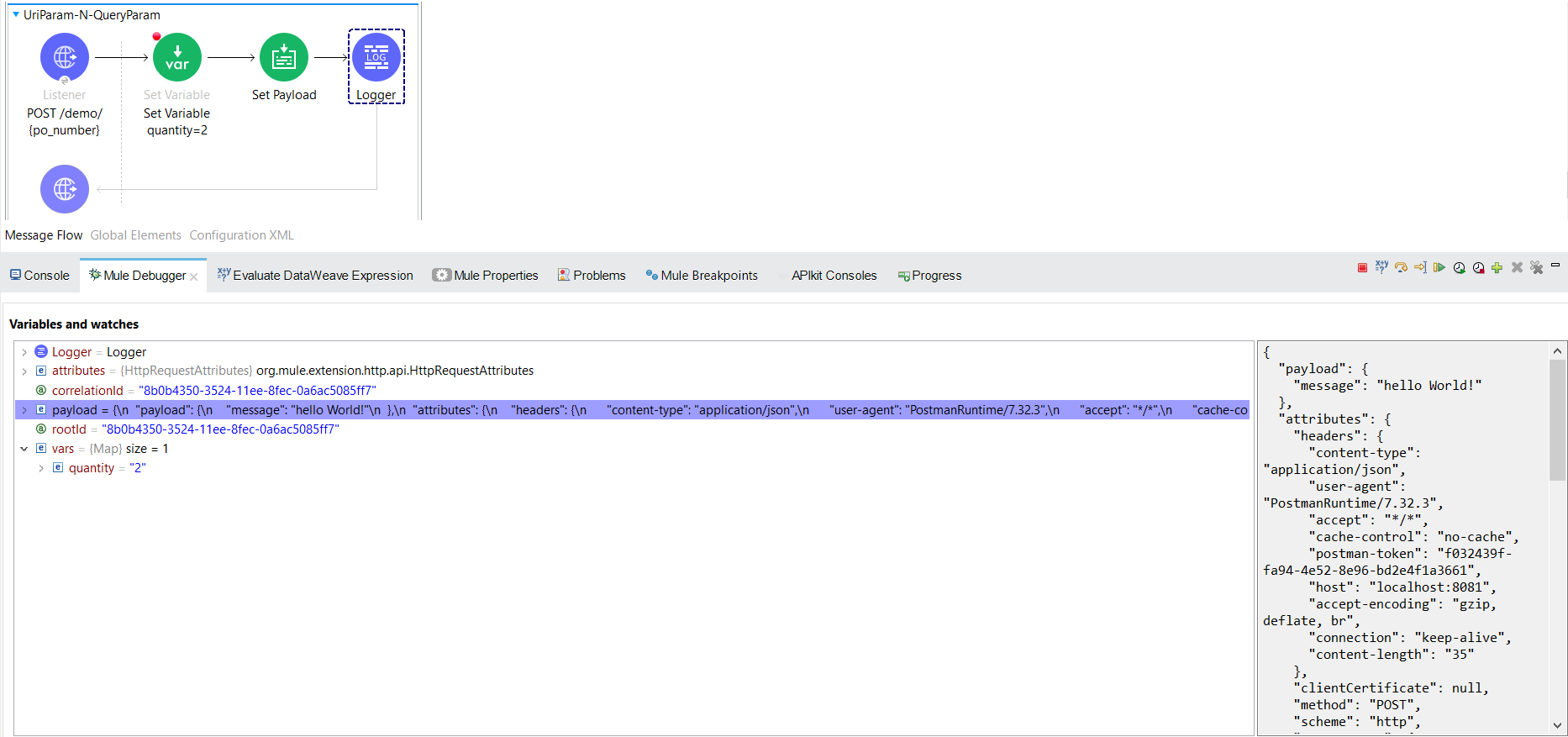
Looking at the above RAML, the URL to get employee details having designation SSE will be GET /employees?designation=SSE.

In the above example, query parameter designation is mandatory and even you can make it optional by modifying the required to false.

## Final RAML With URI Parameter and Query Parameter

|  |
| --- |
| #%RAML 1.0  title: Employees  /employees:  get:  queryParameters:  designation:  description: Desgnation of employee  type: string  required: true  responses:  200:  body:  application/json:  example: |  {"employees":[  {"employeeID":"1", "employeeName":"Tom Berry","designation":"SSE"},  {"employeeID":"2", "employeeName":"JAck Jones","designation":"SSE"}  ]}  /{employeeID}:  description: Return employee information on basis of EmployeeID  get:  responses:  200:  body:  application/json:  example: |  {"employeeID":"1", "employeeName":"Tom Berry","designation":"SSE"} |







Logging Level: Used to filter logs.

ALL < TRACE < DEBUG < INFO < WARN < ERROR < FATAL < OFF

-OFF is the highest level of logging level.This OFF level is used to turn off logging and is the greatest possible rank. With this log level, nothing gets logged at all.

-FATAL means that the application is about to stop a serious problem or corruption from happening. The FATAL level of logging shows that the application’s situation is catastrophic, such that an important function is not working. For example, you can use FATAL log level if the application is unable to connect to the data store-Unlike the FATAL logging level, error does not mean your application is aborting. This log level is used when a severe issue is stopping functions within the application from operating efficiently. Most of the time, the application will continue to run, but eventually, it will need to be addressed.

- The WARN log level is used when you have detected an unexpected application problem. This means you are not quite sure whether the problem will recur or remain. You may not notice any harm to your application at this point. This issue is usually a situation that stops specific processes from running. Yet it does not mean that the application has been harmed. In fact, the code should continue to work as usual. You should eventually check these warnings just in case the problem reoccurs.

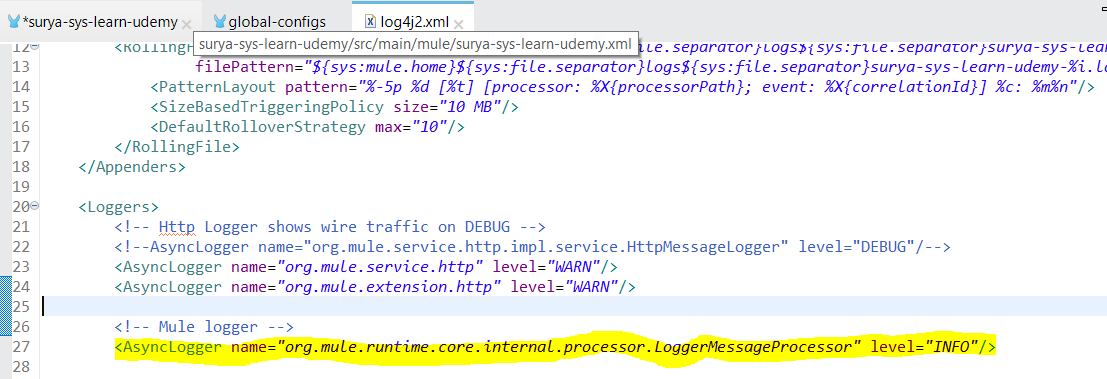
-INFO messages are like the normal behavior of applications. They state what happened. For example, if a particular service stopped or started or you added something to the database. These entries are nothing to worry about during usual operations. The information logged using the INFO log is usually informative, and it does not necessarily require you to follow up on it.

-With DEBUG, you are giving diagnostic information in a detailed manner. It is verbose and has more information than you would need when using the application. DEBUG logging level is used to fetch information needed to diagnose, troubleshoot, or test an application. This ensures a smooth running application.

-The TRACE log level captures all the details about the behavior of the application. It is mostly diagnostic and is more granular and finer than DEBUG log level. This log level is used in situations where you need to see what happened in your application or what happened in the third-party libraries used. You can use the TRACE log level to query parameters in the code or interpret the algorithm’s steps.

-All This log level logs any logging levels that are defined. It logs everything and includes custom logging levels as well. It is the combination of all other logging levels.

-Let say @mule-app level we have defined logger level as Info i.e. in log4j2.xml n in app we have loggers placed at different places with diff level like some are info n some are debug n some are at trace level n some are at Error level.



So if we ran the app then in console we can see logs messages which are @Info n @Error level.this is becoz default we set at info so it will show all logs of logger for which we have specified log level as INFO < WARN < ERROR < FATAL < OFF .

Max-Concurrency:

-max-concurrency applies limit on processing number of messages at a time.

-When a producer component outputs information at a higher rate than a consumer component can process it, you can use the maxConcurrency parameter to set the maximum number of concurrent messages that a flow can process at any given time, whether the components are internal or external, flows or sources, and so on.

-Although Mule applies back-pressure on its own, the maxConcurrency parameter helps to control the flow performance, especially when a backend server may affect when having high concurrency. See how to set the parameter in the following example:

<flow name="maxConcurrency" maxConcurrency="${flow.max.concurrency}">

## Components with Max Concurrency Settings

The following [components](https://docs.mulesoft.com/mule-runtime/latest/about-components) provide a max concurrency setting:

* Flow scope
* Scatter-Gather router
* Async scope
* Batch Job scope
* Parallel For Each scope

Connection-Pooling Profile:

- Mule components configure their component pool which contains multiple instances of the component or connection to process simultaneous incoming requests.

- Connectors such as Anypoint Connector for FTP (FTP Connector) and Anypoint Connector for SFTP (SFTP Connector) pool connections by default and use a pooling-profile to customize the connection pool.

-However, Anypoint Connector for Database (Database Connector) does not perform connection pooling by default, causing every execution of an SQL statement to open a new connection and close it once finished (unless there is a transaction active that uses the same connection for the whole transaction). By using a pooling profile, you enable one-time connection setup and subsequent requests use existing connections, thereby improving request-processing performance:

<db:mysql-config name="dbConfig" host="localhost" port="3306" user="root"

password="" database="esb" doc:name="MySQL Configuration">

<db:pooling-profile maxPoolSize="17" minPoolSize="13" />

</db:mysql-config>

- The maxPoolSize setting specifies the maximum number of connections open to the database, and available to serve incoming requests

Flow,Subflow,Private Flow:

Flow

-is a wrapper of mule components which has source,process n error handling section where source section is used to listen the mule event,process is used to process the mule event n error-handling is used to handle the error.

-A flow without any component inside process section will fail in build phase.

-It is synchronous in nature .To make an flow work as async,use <async /> flow.

-flow or private flow can be called by flow-reference or dataweave lookup function.

**Dataweave lookup function syntax:**

## lookup(flow or PrivatFlowName: String, payload: Any, timeoutMillis?: Number = 2000)

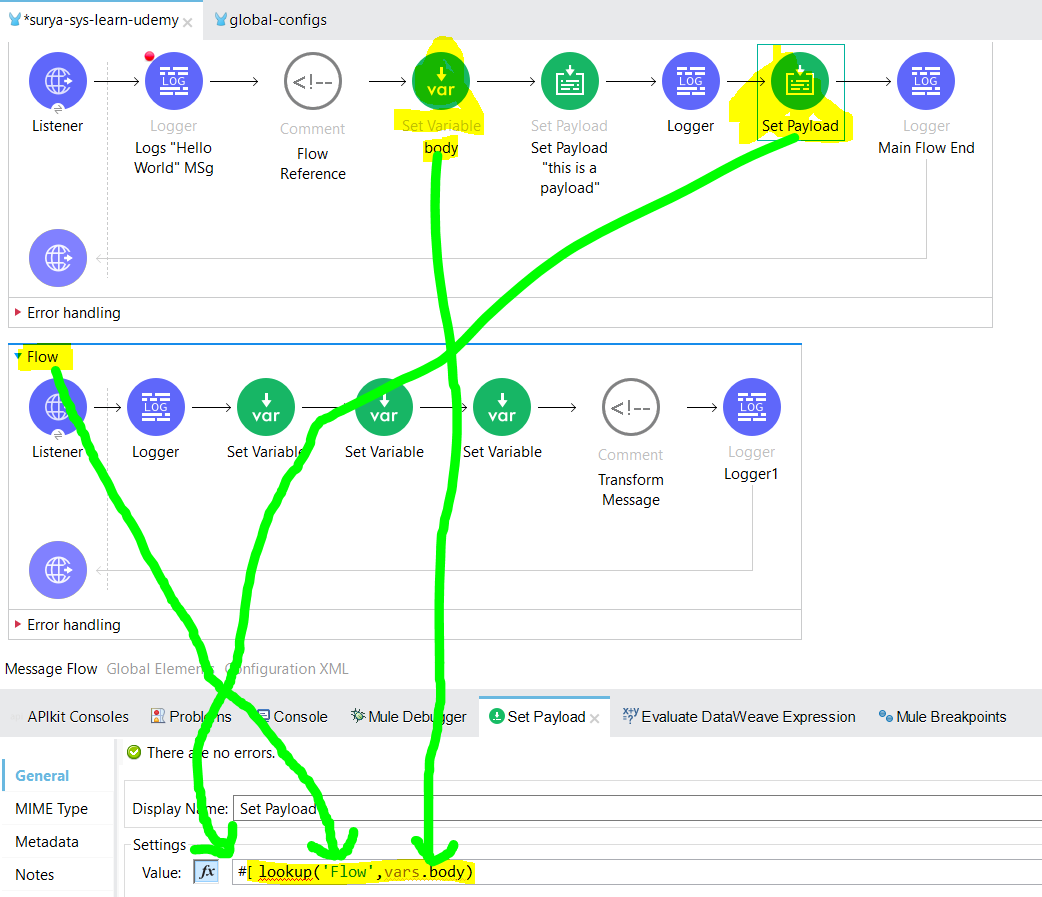
eg: lookup('Flow',vars.body)

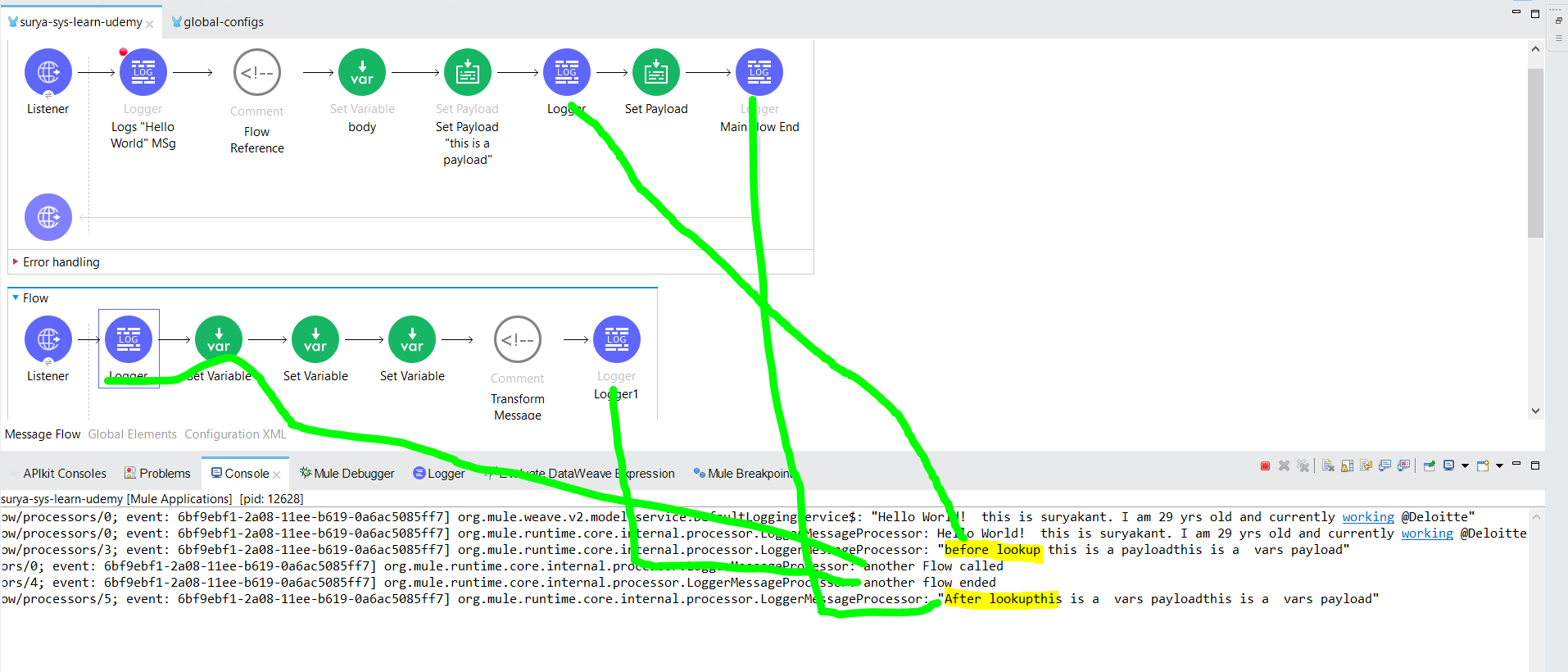
lookup(‘PrivateFlow’,payload)

lookup(‘PrivateFlow’,payload,4000)

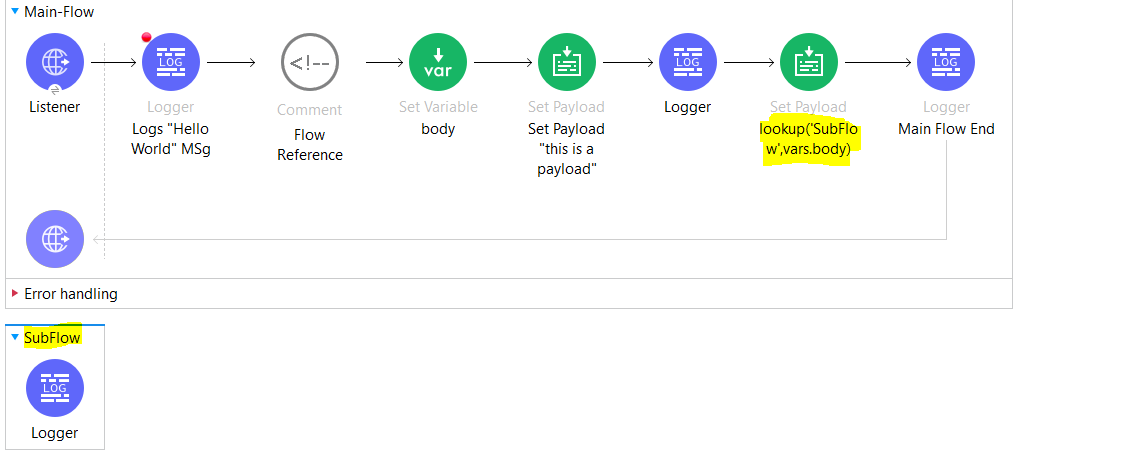
| **Name** | **Description** |
| --- | --- |
| flowName | A string that is name of target flow or Private Flow. |
| payload | The payload to send to the target flow, which can be any (Any) type. |
| timeoutMillis | Optional. Timeout (in milliseconds) for the execution of the target flow. Defaults to 2000 milliseconds (2 seconds) if the thread that is executing is CPU\_LIGHT or CPU\_INTENSIVE, or 1 minute when executing from other threads.  If the lookup takes more time than the specified timeoutMillis value, an error is raised. |

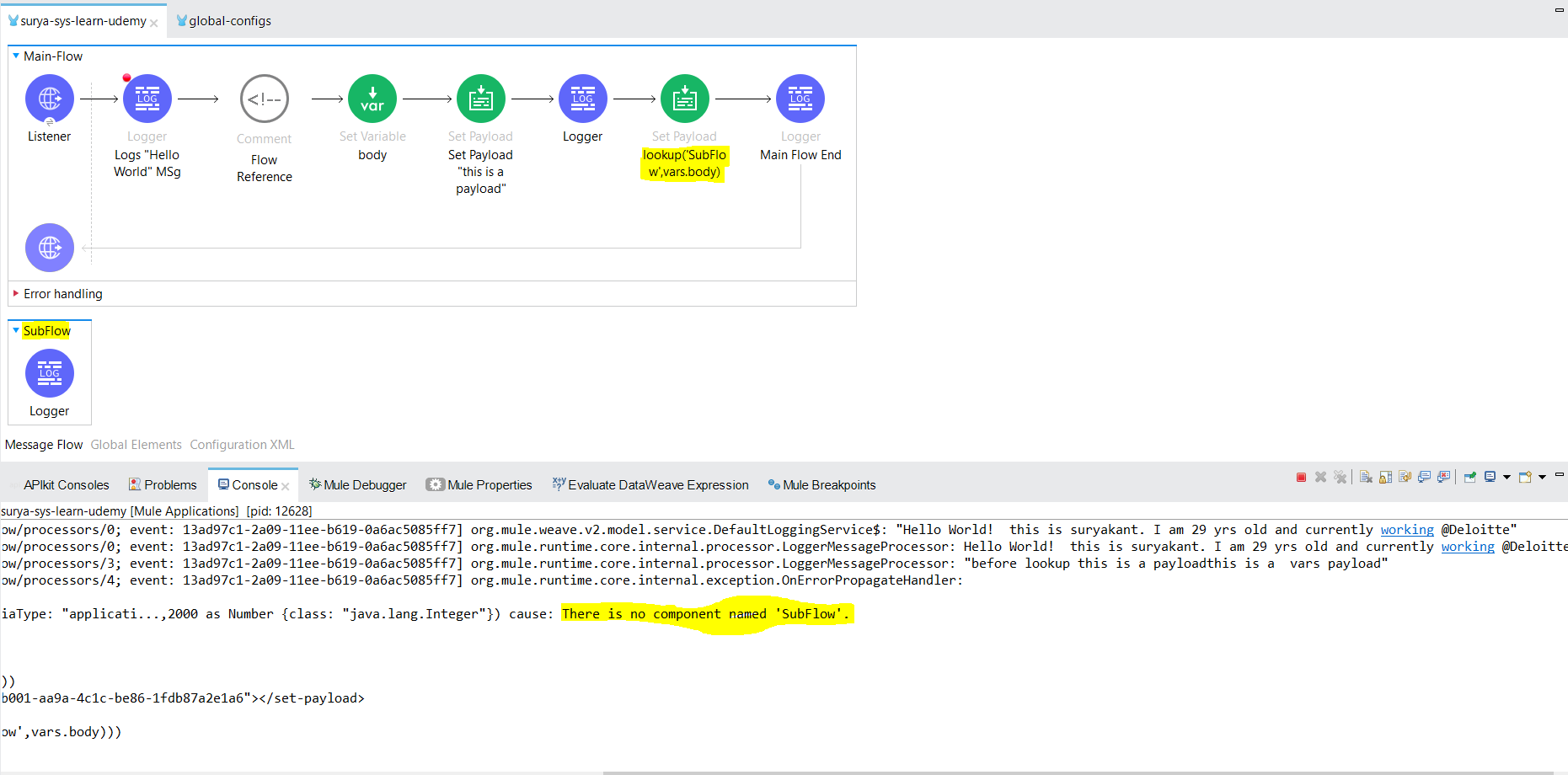
Whatever we provide in Payload section(can be actual payload value or (Variable)vars value). lookup function replaces that as payload n executes the calling flow or private flow with the payload section value provided n returns the payload value after executing complete flow or private flow for further processing.



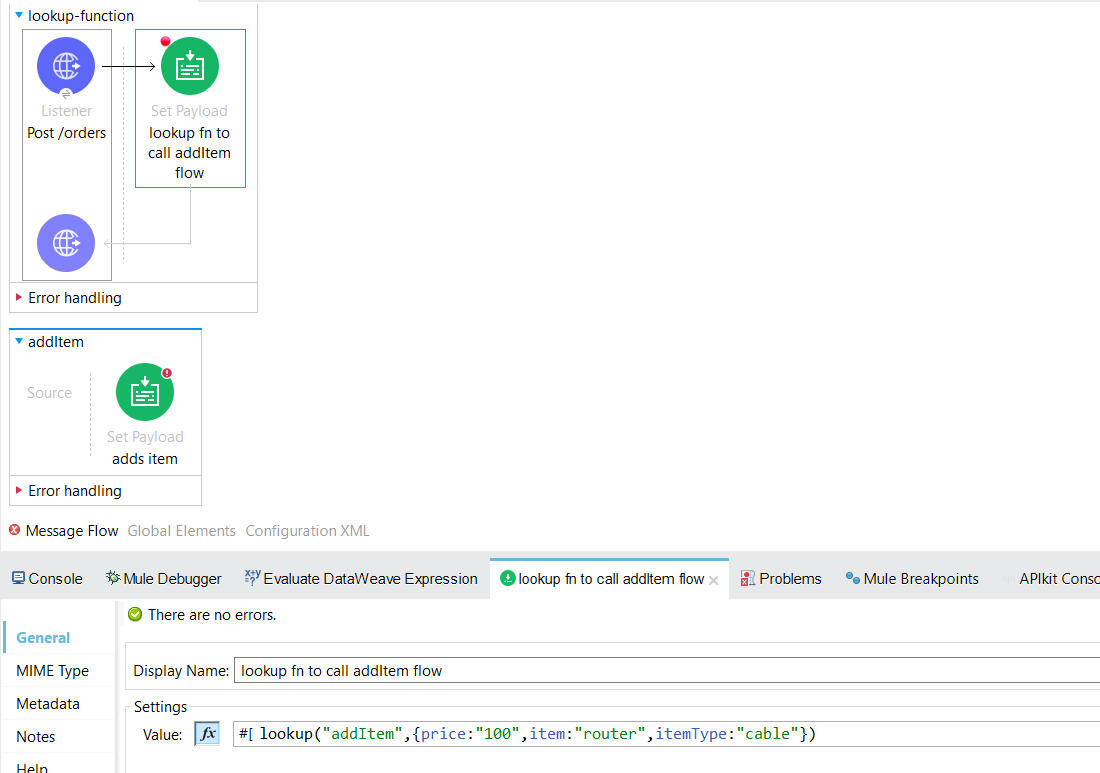


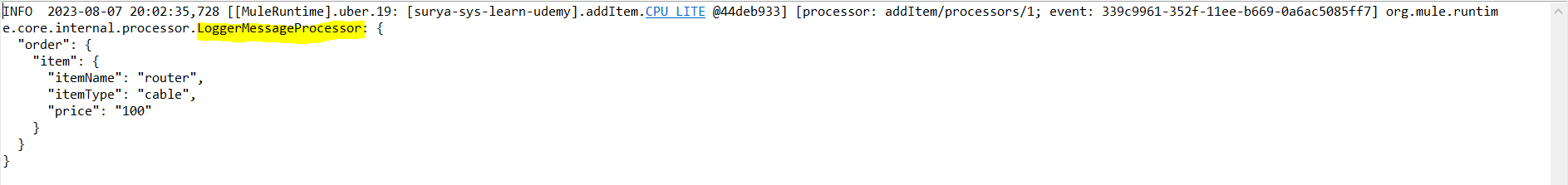
**Note: The lookup function does not support calling subflows. Will get below error**





Exception while executing lookup("SubFlow","this is a vars payload") cause: There is no component named 'SubFlow'.





**Flow-Reference:**

- Flow Reference routes the Mule event to another flow or subflow, executes all processors in the referenced flow, and then routes the event back within the same Mule application.

- When the main flow is triggered, the Mule event travels through and executes the flow until the event reaches the Flow Reference. The event then travels through and executes the referenced flow from beginning to end, and then returns to the main flow.

-This process enables you to treat the entire referenced flow like a single component in your current flow.

**Configuration:**

FlowName: Required.select flow,subflow or private flow name from the drop-down

Target: optional

Target Value: optional

Sometimes you need to execute a flow that uses the current payload and variables, but you want the Mule message to remain unchanged after such process. In this case, you can use a target variable to store the results of the referenced flow processing without changing the original message.

You can store the result of a Flow Reference call in a target variable by configuring the following fields in your Flow Reference properties:

* **Target**: Name of the variable that the Flow Reference creates to store the result of the processed message.
* **Target Value**: An expression that evaluates against the operation’s output. The result of this evaluation is stored in the defined target variable.

-For Flow,we can configure maxconcurrency(at a time how many records gets processed),Initial State(values Started,Stopped [default: Started])

-When we call another flow or subflow or private flow thru flow-reference from main flow it will directly go to process section by ignoring other sections irrespective of flow,subflow,private-flow

Subflow

-is a wrapper of mule components which has only process section in it which is used to process the mule event.it does not have source n error-handler section.

-A subflow without any component inside process section will fail in build phase

-It is synchronous in nature .To make an Subflow work as async,use <async /> flow

-To provide Error Handling section to Subflow, we can use Try Scope <try /> inside Subflow.

-subflow can be called by flow reference only.

Private Flow

-A flow with no source component defined in source section of the flow.

- For Private Flow,we can configure maxconcurrency(at a time how many records gets processed),Initial State(values Started,Stopped [default: Started])

-A Private flow can be called by flow-reference or dataweave lookup function.

Consuming Web Services:

-API(Application Programming Interface) is an interface which allows two systems to communicate with each other.

-Webservices is a way of communication thru which two systems can communicate with each other.There are two types:

1. Rest-Based(Representational State Transfer): uses HTTP methods,Simpler,accepts n provides result in any format like json,xml
2. Soap-Based(Simple Object Access Protocol):uses WSDL file(Web service Description Language),Complex,Accepts n provides result in xml only

To Consume above mentioned webservices,we can use:

1. **HTTP Request( For RESTful Services)**
2. **Web Service Consumer(For Soap-based Services)**

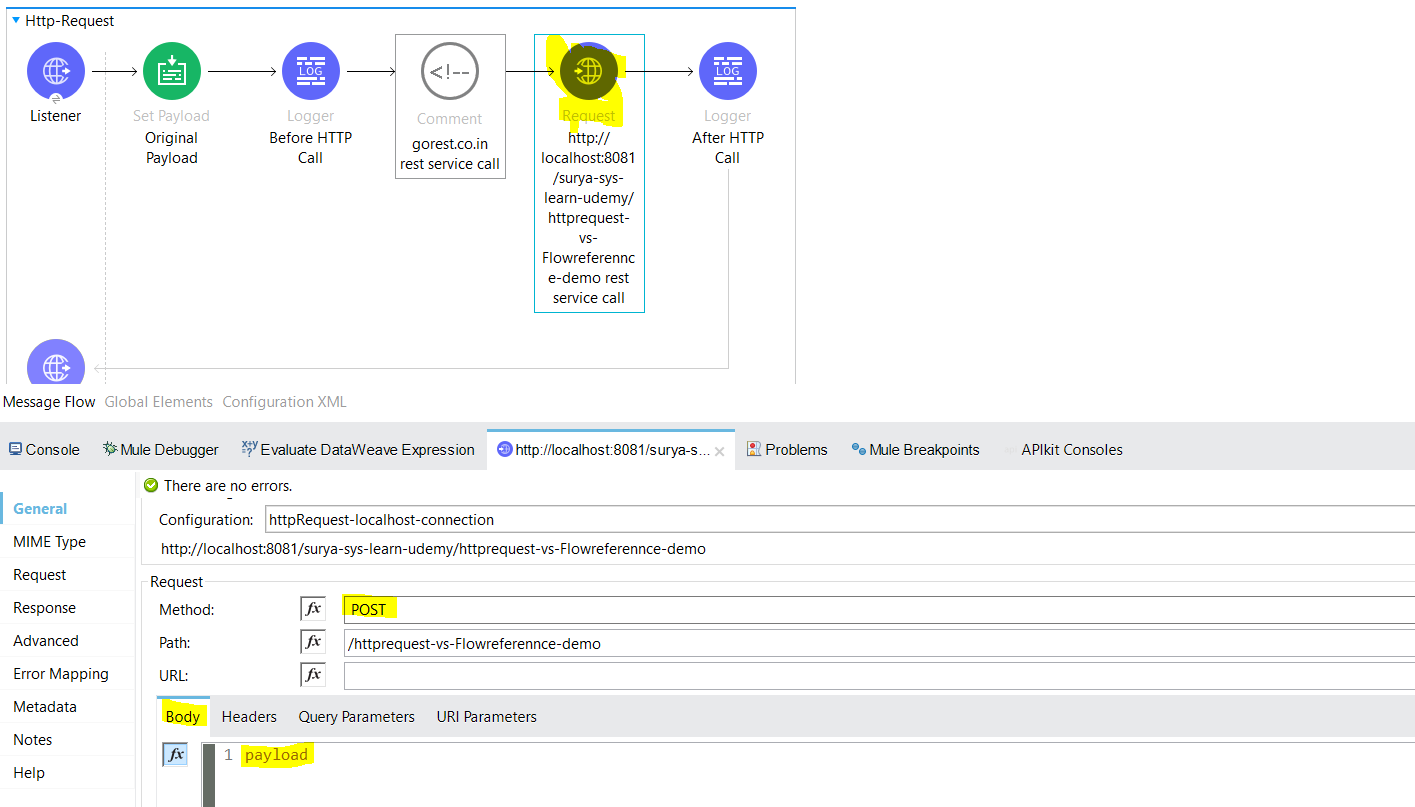
**HTTP Request(RESTful Services):**

**-**To Consume RestFul Webservices we can use HTTP Requester Component of the Mule palette.

-The Anypoint Connector for HTTP (HTTP Connector) **Request** operation enables you to consume an external HTTP service using the default GET method. Otherwise, you can choose what methods the operation accepts:

* The GET and OPTIONS methods do not send the payload in the request (the body of the HTTP request is empty).even if we have body defined as payload by default it will not send payload
* The CONNECT, DELETE, PATCH, POST, and PUT methods send the message payload as the body of the HTTP request.

Example :



-In Http-Requester connector ,If we defined Method as “GET”(screenshot shows POST intentionally) then let say before calling http-requester we have set payload as “Original Payload” and in Http-Requester connector we have provided body as payload like above screenshot then while we ran the program n the runner comes to the Http-Requester n it starts the execution of HTTP Requester it considers body as empty(payload).

Logs:

org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {

"message": "Before HTTP Call",

"payload": "Original Payload"

}

org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {

"message": "http://localhost:8081/surya-sys-learn-udemy/httprequest-vs-Flowreferennce-demo rest service call started",

"payload": ""

}

-In Http-Requester connector ,If we defined Method as “POST” then let say before calling http-requester we have set payload as “Original Payload” and in Http-Requester connector we have provided body as payload like above screenshot then while we ran the program n the runner comes to the Http-Requester n it starts the execution of HTTP Requester it considers body as Original Payload

Logs:

org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {

"message": "Before HTTP Call",

"payload": "Original Payload"

}

org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {

"message": "http://localhost:8081/surya-sys-learn-udemy/httprequest-vs-Flowreferennce-demo rest service call started",

"payload": "Original Payload

}

Payload, Vars, Attributes Accessibility Scope in Case of Flow-Reference and HTTP-Requester:

-Considered below flows for explanation:

Assume we are using 2 flows,main flow n child flow as shown in the screenshot intially before callling flow -reference or http request we have defined paylod as “original Payload” n variable myvar=1 n attributes showing requestPath as ‘/surya-sys-learn-udemy/demo3’ .So we need to check whether we can access these 3 params in child flow as it is or there will be some changes

Case 1: Only Flow reference is used then

-Incase we are using only flow-reference, then we can access payload,vars,attributes in child flow as well + if we want we can modify the payload n vars value as well in child flow n then once we back from child flow to main flow we can get the modified vars n payload(if modified in child flow) to process further.

Logs:

org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {

"message": "Before Flow Reference",

"payload": "Original payload",

"myvar": "1",

"attributes": {

"requestPath": "/surya-sys-learn-udemy/demo3"

}

}

org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {

"message": "http://localhost:8081/surya-sys-learn-udemy/httprequest-vs-Flowreferennce-demo rest service call started",

"payload": "Original payload",

"myvar": "1",

"attributes": {

"requestPath": "/surya-sys-learn-udemy/demo3"

}

}

org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {

"message": "http://localhost:8081/surya-sys-learn-udemy/httprequest-vs-Flowreferennce-demo rest service call ended",

"payload": "modified payload",

"myvar": "2",

"attributes": {

requestPath": "/surya-sys-learn-udemy/demo3"

}

}

org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {

"message": "After Flow Reference",

"payload": "modified payload",

"myvar": "2",

"attributes": {

"requestPath": "/surya-sys-learn-udemy/demo3"

}

}

Case 2: Only Http-Requester is used then

-Incase we are using only http-requester, then we can access only the payload in child flow, vars defined at main flow wont be accessible inside child flow but they can be accessible once child flow completes and the trigeer goes to main flow( the value set at main flow will be visible) n old attributes will be replaced by http requesters attributes+ if we want we can modify the original payload from main flow in child flow n the modified payload is accessible at main flow once trigger goes from child flow to main flow again n if we want we can define vars at child level but these variables will be accessible till child flow completes once the trigeer goes to main flow we cant access the child variables in main flow .

Logs:

org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {

"message": "Before Flow Reference",

"payload": "Original payload",

"myvar": "1",

"attributes": {

"requestPath": "/surya-sys-learn-udemy/demo3"

}

}

org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {

"message": "After Flow Reference",

"payload": "Original payload",

"myvar": "1",

"attributes": {

"requestPath": "/surya-sys-learn-udemy/demo3"

}

}

org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {

"message": "Before HTTP Call",

"payload": "Original payload",

"myvar": "1",

"attributes": {

"requestPath": "/surya-sys-learn-udemy/demo3"

}

}

org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {

"message": "http://localhost:8081/surya-sys-learn-udemy/httprequest-vs-Flowreferennce-demo rest service call started",

"payload": "",(Here payload is showing as “”[empty string] becoz we configured GET method in http-requester connector n in body passed payload as standard GET method doesn’t need request-body in a call)

"myvar": "",

"attributes": {

"requestPath": "/surya-sys-learn-udemy/httprequest-vs-Flowreferennce-demo"

}

}

org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {

"message": "http://localhost:8081/surya-sys-learn-udemy/httprequest-vs-Flowreferennce-demo rest service call ended",

"payload": "modified payload",

"myvar": "2",(this variable is used/declared in child flow,so only accessible inside child flow myvar= 2)

"attributes": {

"requestPath": "/surya-sys-learn-udemy/httprequest-vs-Flowreferennce-demo"

}

}

org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {

"message": "After HTTP Call",

"payload": "modified payload",

"myvar": "1", (the original myvar=1 declared/used at main flow,will be used again after the child flow completes)

"attributes": {

"headers": {

"content-length": "16",

"date": "Tue, 25 Jul 2023 07:20:33 GMT"

},

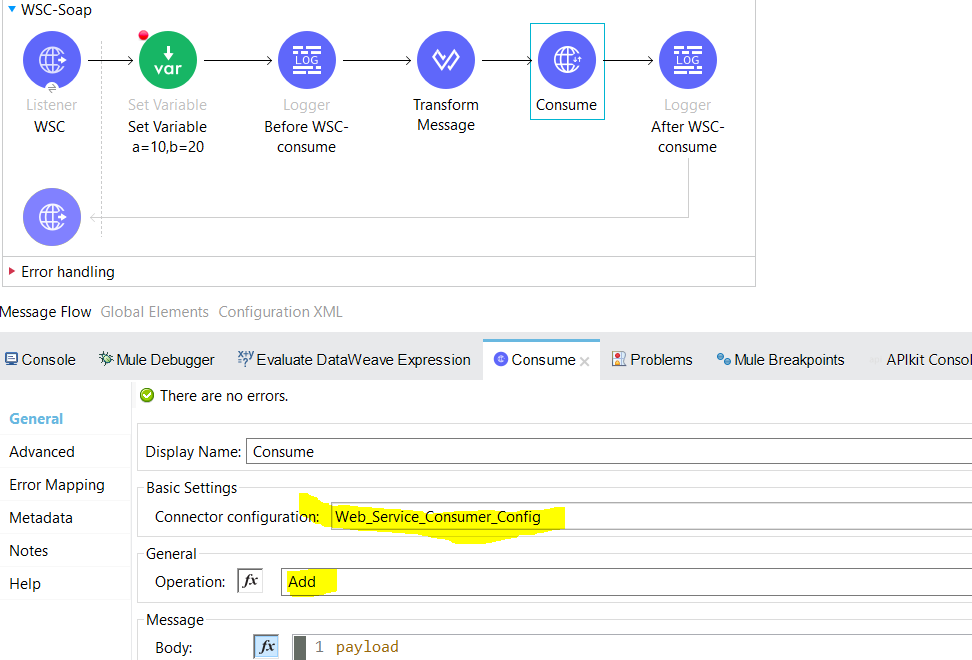
"reasonPhrase": "",

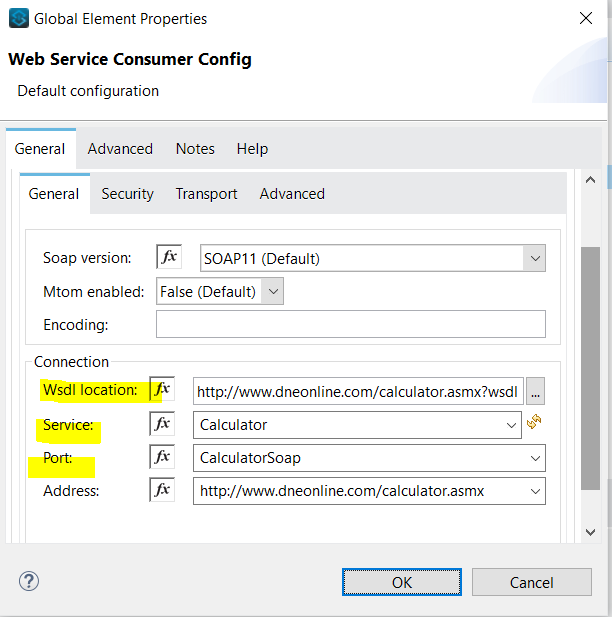
"statusCode": 200

}

}

**Web Service Consumer WSC(Soap-Based Services):**





-To Consume SOAP Webservices,we can use Web Service Consumer Component of the Mule palette.

-WSC global config has below parameters :

**WSDL location (Required)**  
The remote or local WSDL file URL. WSDL, or Web Service Description Language, is an XML based definition language. It’s used for describing the functionality of a SOAP based web service. WSDL files define various aspects of SOAP messages:

* Whether any element or attribute is allowed to appear multiple times
* The required or optional elements and attributes
* A specific order of elements, if it is required
* It tells about the Endpoints,Request/Response Structure.

Note: As RESTFul API uses RAML API Specifications which is a contract between Rest API Provider and the Rest API Consumer defines the endpoint,req/response structure of REST API same WSDL file is a contract between the SOAP API provider and the SOAP API consumer of the service which defines the endpoint,req/response structure of SOAP API .

* **Service (Required)**  
  The service name
* **Port (Required)**  
  The port name
* **Address**  
  The address of the web server to dispatch requests if the **Service** and **Port** parameters did not automatically provide this value.

-It has only one operation called as ‘Consume’.Consume operation is used to Consume an operation from a SOAP Web Service.

-Required Properties for Consume Operation :

* **Operation(Required)**  
  Defines which SOAP operation of the web service to invoke. During the design phase, the parameter defines the input and output types for the Consume operation. The types will change depending on which operation you choose.
* **Message**  
  A representation of SOAP:ENVELOP composed of three optional parameters:
  + **Body**  
    The XML body to include in the SOAP message, with all the required parameters, or null if no parameters are required
* The default value is #[payload], based on the assumption that the incoming payload is the XML entity ready to ship to the service.
* If the body is not valid XML, or if the request cannot be created for some reason, you get a WSC:BAD\_REQUEST error.
* If you don’t provide body content, the Web Service Consumer Connector attempts to generate one, and this works only for cases where no XML entity is expected in the body.
* Some web services require to append the XML prolog tag into the envelope’s body XML content, which contains the version and encoding information that identifies the document as being XML.
  + **Headers**  
    The XML headers to include in the SOAP message
  + **Attachments**  
    The attachments to include in the SOAP request

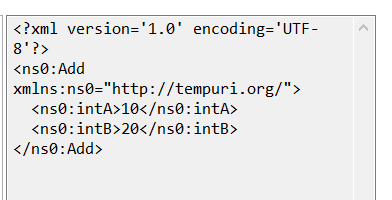
Output:

The output of the Consume operation represents an incoming SOAP message that contains the same elements that the Message parameter has, and you can access each part of it.

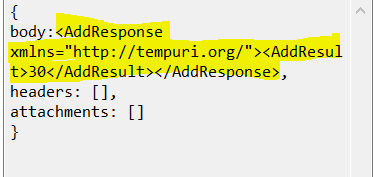
* + - Body: can be accessed by payload.body
    - Headers: can be accessed by payload.headers
    - Attachments: can be accessed by payload.attachment

Note: SOAP accepts input in XML n returns the response in XML

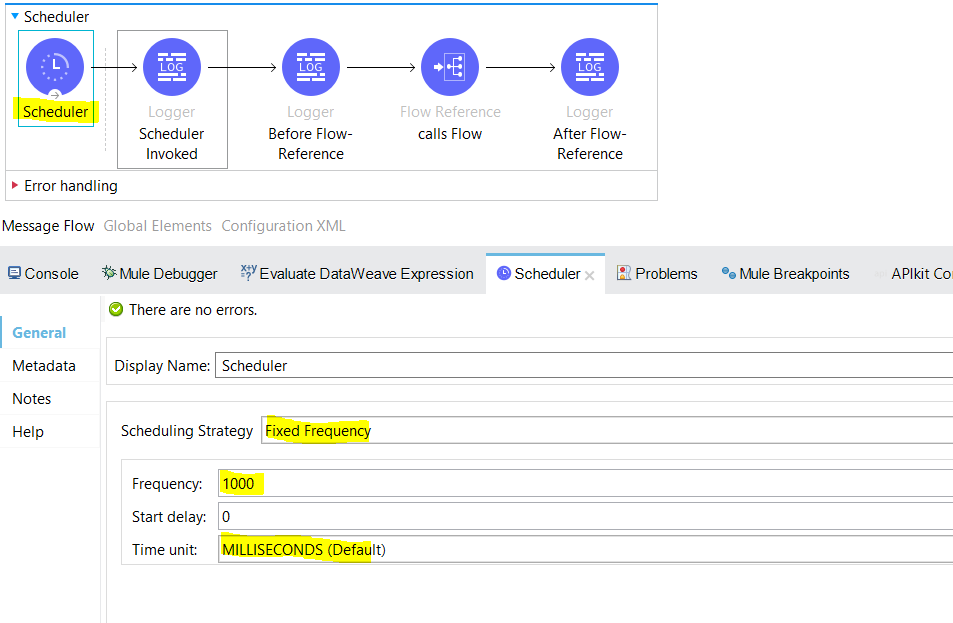
Request:



Response:



Scheduler Component: It is used to trigger the flow based on time condition.



-It is a part of Source Section that schedules periodic execution of a flow. By default it will used a fixed frequency scheduling of 1 second.

- The Scheduler component is a Mule event source that triggers the execution of a flow based on a time-based condition. For example, a Scheduler might create and send a message that will trigger the flow processing every 5 seconds or every day at 12:00 in a given time zone.

Consider the following when adding a Scheduler to your Mule application:

* Schedulers use the same timezone as the machine on which Mule is running. However, if an application is running in CloudHub, the Scheduler conforms to the UTC timezone, regardless of the geographic region in which the application is running.

To set a different time zone for the Scheduler, see timeZone in [Cron (<cron/>)](https://docs.mulesoft.com/mule-runtime/latest/scheduler-concept#props_cron_expressions).

* The disallowConcurrentExecution property enables you to prevent the Scheduler from triggering the execution of the flow while a previously-triggered execution is running.

Note: By default in Scheduler Component we cannot able to see placeholder for disallowConcurrentExecution property .To define n to set the value(true/false) for disallowConcurrentExecution property we should go to “ConfigurationXML” Section on canvas ->go to scheduler component declaration tag <scheduler > n type dis ,press ctrl+ space bar ,it will autofill the property name to disallowConcurrentExecution then set the value for it (true/false).By default it is false which means the Scheduler does not wait for one execution of a given flow to complete before executing another instance of the flow at the configured cadence. You can change this behavior by setting the disallowConcurrentExecution="true" attribute directly in the configuration XML.

| **Attribute XML** | **Description** |
| --- | --- |
| disallowConcurrentExecution | When this attribute is set to true, the Scheduler skips its scheduled execution if a previous execution is still running. Once the flow finishes, the Scheduler executes at the next configured interval. The default value is false. |

In the following example, the Scheduler skips its execution if the flow takes longer to complete than the fixed frequency of 10 milliseconds:

<flow name="scheduler-disallowConcurrentExecution-ex" >

<scheduler doc:name="Scheduler" disallowConcurrentExecution="true" >

<scheduling-strategy>

<fixed-frequency frequency="10" timeUnit="MILLISECONDS"/>

</scheduling-strategy>

</scheduler>

<!-- processors here -->

</flow>

Each time a Scheduler skips its execution, Mule logs an Execution skipped message, for example (edited for readability):

INFO 2022-11-09 15:15:43,082 ...

...scheduler.DefaultSchedulerMessageSource:

Flow 'scheduler-disallowConcurrentExecution-ex' is already running and

'disallowConcurrentExecution' is set to 'true'. Execution skipped.

INFO 2022-11-09 15:15:43,083 ...

...scheduler.DefaultSchedulerMessageSource:

Flow 'scheduler-disallowConcurrentExecution-ex' is already running and

'disallowConcurrentExecution' is set to 'true'. Execution skipped.

* If [back-pressure](https://docs.mulesoft.com/mule-runtime/latest/execution-engine#backpressure) occurs because no resources are available at the time of the scheduled trigger, Mule skips that execution.
* By Default If we drag Scheduler into the canvas then we get Scheduling strategy as Fixed Frequency with 1000msec frequncy and start delay as 0 value, which means scheduler will auto trigeer a flow after every 1 second.
* In Fixed Frequency we have time unit as Milliseconds(Default),Seconds,Minutes,Hours,Days

### If you want to set scheduler to run @ specific date n time then we can use CRON expressions

### Cron (<cron/>)

For more complex scheduling strategies, you can use a cron expression.

| **Attribute** | **XML** | **Description** |
| --- | --- | --- |
| **Expression** | expression | A cron expression for triggering the Scheduler. There is no default cron expression. For guidance with expressions, see [Cron Expressions](https://docs.mulesoft.com/mule-runtime/4.4/scheduler-concept#cron_expressions). Example: expression="1 1 1 1,7 \* ?" |
| **Time Zone** | timeZone | Time zone passed as a system property, or in a machine’s operating system. [Java time zone](https://docs.oracle.com/javase/7/docs/api/java/util/TimeZone.html) values are supported. Avoid Java abbreviations, such as PST and AGT, and instead use the full-name Java equivalents, such as America/Los\_Angeles. Example: timeZone="America/Argentina/Buenos\_Aires" |

The following example uses a cron expression to trigger the flow at 12:00 every day in the America/Los\_Angeles time zone.

<flow name="componentsFlow">

<scheduler>

<scheduling-strategy>

<cron expression="0 0 12 \* \* ?" timeZone="America/Los\_Angeles"/>

</scheduling-strategy>

</scheduler>

<logger message="my message"/>

</flow>

XML✓ Copied

## Cron Expressions

Cron is a widely used standard for describing time and date information. The Cron Expression (<cron expression /> scheduling strategy (<scheduling-strategy >) is useful for triggering a flow at intervals not available through the Fixed Frequency scheduling strategy.

The Scheduler keeps track of every second and creates a Mule event when the Quartz Cron expression matches your time-date setting. You can trigger the event just once or at regular intervals.

A date-time expression consists of six required settings and can include the optional year setting. Specify the settings in the following order:

1. Seconds (0-59)
2. Minutes (0-59)
3. Hours (0-23)
4. Day of month (1-31)
5. Month (1-12 or JAN-DEC)
6. Day of the week (1-7 or SUN-SAT)
7. Year (empty or a 4-digit year between 1970-2099, for example, 2019)

The Scheduler supports Quartz Cron expressions. Here are a few examples:

| **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. |

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.

This example logs the message "hello" every second:

<flow name="cronFlow" >

<scheduler doc:name="Scheduler" >

<scheduling-strategy >

<cron expression="\* \* \* \* \* ?" />

</scheduling-strategy>

</scheduler>

<logger level="INFO" doc:name="Logger" message='"hello"'/>

</flow>

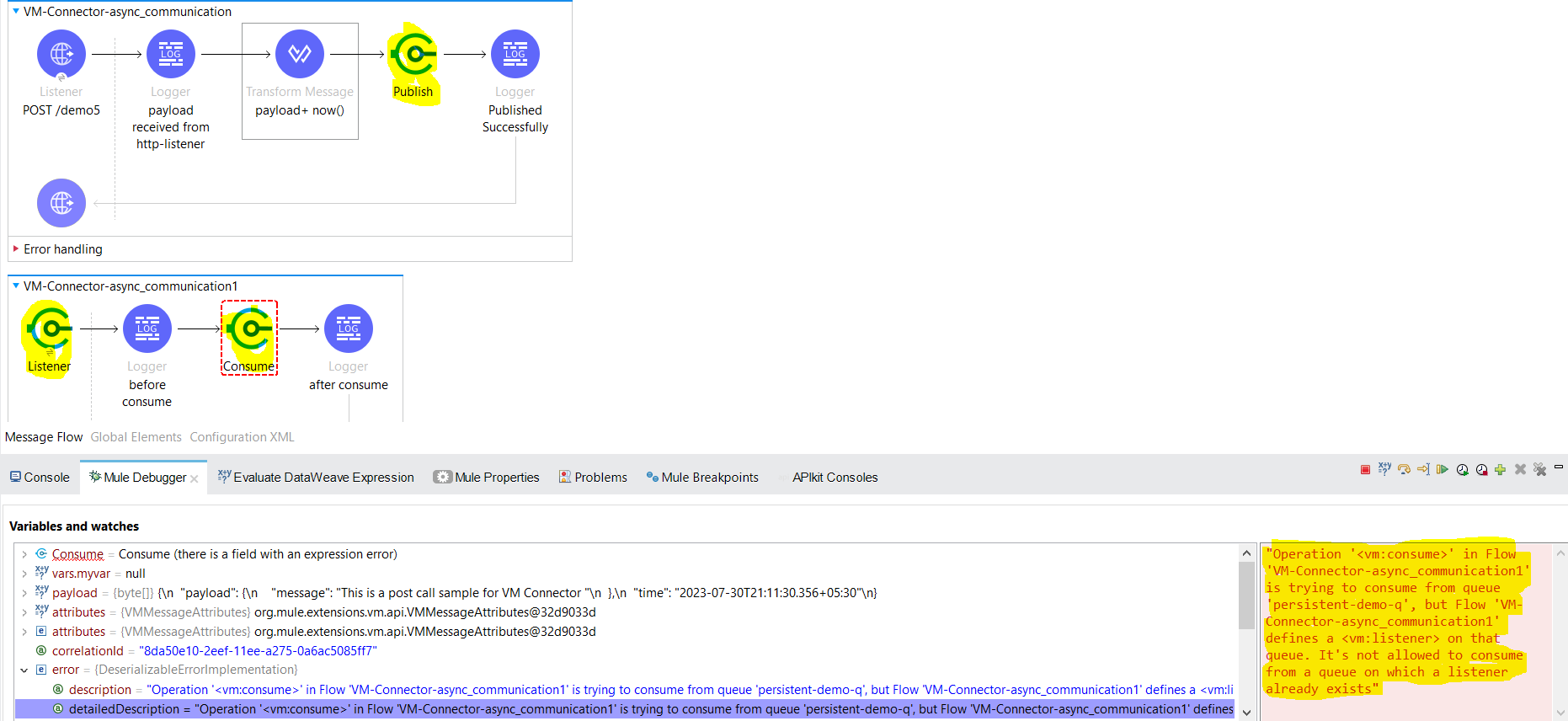
VM Connectors(Pub-Sub integrations):

-VM Connectors (Virtual Machines) Connectors are used for inter-app(one mule app-to another mule app if they are developed in same domain) and Intra-app (one flow to another flow in same mule app) provides async(publish,listener,consume opertaion)as well as sync communications(publish-consume operation).

-In VM config,we can define two types of queue

* Persistent Queue- are slower becoz msgs stored in disk so consumes time but reliable as they can tackle system crash or restart n non-volatile that means msgs does not get lost in case of system crash or restart as they store msg in disk.
* Transient Queue- are faster bcoz msgs stored in memory n they are not reliable also which means tey cannot tackle system crash or restart n they are volatile.

-We cant use VM Listener if we are using publish-consume operation or we are using consume operation to consume the msg from the same queue as VM Listener.otherwise we will get below error.



-Publish-consume is used for sync communications,it first publishes the msg n wait for the assigned time to get the msg consumed at the publish-consume operation if not consumed then throws queue timeout.

-publish,consume,listener are used for async communications.publish is used for publishing the msg .if unable to publish the msg within assigned timeout time then throws queue timeout error. consume is used for consumeing the msg .if unable to consume the msg within assigned timeout time then throws queue timeout error.

Note: Note that persistent queues are unavailable in CloudHub 2.0.

File Connector:

 -manages files and folders on a locally mounted file system. The connector’s main features include:

* The ability to read files or fully list directory contents on demand
* Support for common file operations such as creating directories, reading, writing, copying, moving, renaming, deleting and listing files

File operations:

**On New or Updated File**Initiates a flow by polling a directory for files that have been created or updated and generates a message for each file found. [Trigger a Flow for a Newly Created or Updated File](https://docs.mulesoft.com/file-connector/1.5/file-on-new-file). Configure the **On New or Updated File** source to trigger a flow for a new created or modified file.It has scheduling strategy (fixed or cron) which can be applied to have a check for a file in periodical manner. Anypoint Connectors for File, FTP, and SFTP provide an **On New or Updated File** operation that polls a directory for files that have been created or updated. A message is generated for each file that is found. The following strategies determine that a file is new:

* Set the Auto delete field to True to delete each file after it is processed, which causes all files obtained in the next poll to be considered new.
* Set the Move to directory field to move each file to a different directory after it is processed, which achieves the same effect as Auto delete but without erasing the file.
* Set the Watermark field to pick only files that have been created or updated after the last poll was executed.

You can also use a matcher for additional filtering of files.

File Listener (On New File Trigger)

<flow name="onNewFile">

<file:listener config-ref="file" directory="test-data/in" autoDelete="true">

<scheduling-strategy>

<fixed-frequency frequency="1000"/>

</scheduling-strategy>

</file:listener>

<flow-ref name="processFile" />

</flow>

## Watermarking

For the File connector, the watermarking strategy can be used in two different modes: CREATION\_TIMESTAMP or MODIFIED\_TIMESTAMP. The strategy depends on whether you want to filter files based on their creation or modification time.

Example: Watermarking with the File Listener

<file:listener doc:name="On New File"

config-ref="File\_Config1"

watermarkMode="CREATED\_TIMESTAMP">

<scheduling-strategy >

<fixed-frequency />

</scheduling-strategy>

</file:listener>

[List Files](https://docs.mulesoft.com/file-connector/1.5/file-list)  
Configure the **List** operation to list the files and folders in the configured directory path.

Use the List operation to list the files and folders in the path pointed to by the directoryPath parameter. The List operation returns an array of messages in which:

Each message holds the file’s content in its payload.

The file’s message attributes section carries the file’s metadata (such as name, creation time, and size).

The payload is empty if the element is a folder.

The List operation requires the directoryPath parameter, which represents the relative path of the directory to list, unless you specify the full path of the directory. The path is relative to the working directory value defined in the configuration referenced by the config-ref parameter. If no configuration is referenced, the working directory defaults to the value of the user.home system property. If the system property is not set, the connector fails to initialize.

[Create a Directory](https://docs.mulesoft.com/file-connector/1.5/file-create-directory)  
Configure the **Create directory** operation to create a directory of a given name.

Anypoint Connector for File (File Connector) provides a **Create directory** operation that enables you to create a directory of a given name.

If the reason for creating the directory is to immediately write, copy, or move content to the directory, then use the **Write**, **Copy**, or **Move** operations with the **Create parent directories** field set to **TRUE**.

[Read a File](https://docs.mulesoft.com/file-connector/1.5/file-read)  
Configure the **Read** operation to read a file at any point in the flow. the only required parameter for the Read operation is the path to the file to be read. The path can be either the full path to the file or a relative path. The path uses the working directory value defined in the configuration referenced by the config-ref parameter. In Anypoint Studio, you can set the path in the **File path** field. In XML, use the **path** attribute in the <file:read statement to specify the path.

If no configuration is referenced, the working directory defaults to the value of the user.home system property. If the system property is not set, the connector fails to initialize. The Read operation returns a message with:

* The file’s content as the payload
* The file’s metadata in the message attributes, metadata (such as name, creation time, size, and so on) in the message attributes.

If the file does not exist, the Read operation throws a FILE:ILLEGAL\_PATH error. Note that the operation does not read directories.

## Configure a Relative Path

If numerous Read operations (or any other file operations) in the same Mule app share part of their path, their base path can be extracted into a [file configuration](https://docs.mulesoft.com/file-connector/1.5/#connection_settings) for better reuse.

For example, to access two files whose full paths are /Users/Documents/Examples/manual.txt and /Users/Documents/Examples/readme.txt, you can define the part of the path that is the same in a configuration, for example:

<file:config name="File\_Config" >

<file:connection workingDir="/Users/Documents/Examples" />

</file:config>

The resulting Read operations that reference the configuration would be:

<flow>

<file:read config-ref="File\_Config" path="manual.txt" />

<file:read config-ref="File\_Config" path="readme.txt" />

</flow>

[Write a File](https://docs.mulesoft.com/file-connector/1.5/file-write)  
Configure the **Write** operation to write content into the given path on demand.

A Write operation is available to the File, FTP, and SFTP connectors. For these connectors, the operation writes content into the given path on demand. It supports common use cases described below.

Embedded DataWeave Transformations

By default, the connector writes whatever is in the message payload:

Example: File Connector

<file:write path="output.csv" />

However, if the payload is a different format (for example, not CSV) and you need to transform it before writing it, what do you do? If you place a Transform component before the Write operation, the message payload changes and that impacts the operation that is placed after the Write operation.

To avoid this undesired impact, you can place the transformation inside the Write operation to generate content that will be written without producing a side effect on the message in transit.

The next example uses the File connector to write the content:

Example: Transformation within a File Write Operation

<file:write path="output.csv">

<file:content>#[%dw 2.0

output application/csv

---

payload.customers.email

]

</file:content>

</file:write>

XML✓ Copied

Writing into Directories

If any of the a, b, or c directories in the example below do not exist, the Write operation will fail by default. However, if you set the createParentDirectories to true, the connector will automatically create any missing directories.

Example: File Connector writes to a Directory

<file:write path="a/b/c/myFile.txt" />

## Write to Existing Files

When writing content to a file, set **Write Mode** to any of the following modes:

* **OVERWRITE (Default)**  
  If the file exists, overwrite it completely.
* **APPEND**  
  If the file exists, add its content to the end of an existing file.
* **CREATE\_NEW**  
  Create a new file. If the file already exists, then you receive an error.

## Locks

The **Write** operation supports locking similar to the **Read** operation. The main difference is that the lock releases automatically after the **Write** operation finishes. Set **Lock** to any of the following modes:

* **TRUE**
* **FALSE (Default)**
* [Copy and Move Files](https://docs.mulesoft.com/file-connector/1.5/file-copy-move)  
  Configure the **Copy** and **Move** operations to copy or move files or directories on demand.

Anypoint Connector for File (File Connector) provides the Copy and Move operations that enable you to copy or move files or directories on demand.

* Source path  
  The directory path name or file name to copy or move. If a file or directory already exists and the Overwrite field is set to false, the connector throws the message FILE:FILE\_ALREADY\_EXISTS.
* Target path  
  The directory path name into which to copy or move the file. If a target path doesn’t exist, and neither does its parent, the connector attempts to create the path depending on the value of the Create parent directories argument. If Create parent directories is false, then FILE:ILLEGAL\_PATH occurs.
* Rename to  
  Renames the file as part of the operation. Set this field with a file name instead of a path. The connector retains the original file name if you do not configure this field.
* Delete File:

This operation deletes the file.

For-Each(Synchronous Processing):

-For Each scope splits the payload into elements n processes them one by one thru the componenents tht u place in the scope.It can process any collection like lists,array etc.

-The For Each scope is similar to a "foreach" or "for loop" in most programming languages. It takes as input a "collection" (a list or array from a payload with any kind of content), and iterates over the items in the collection, passing each one through the components placed inside the For Each scope. If a particular item in a collection throws an exception, the For Each scope will stop processing it and invoke an error handler.

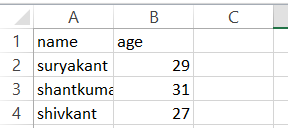
-For-each is single-threaded

-Incase of exception scenarios foreach stops processing the records as soon as it detects some error n all the previous records which got successful will be successfully processed.

Foreach config: batchsize=3 n in payload passed array of 4 objects so in 1st iteration it will write 3 object in file n in 2nd iteration it will raise error as sizeOf(payload) !=3 it is actually 1



Output: since sizeOf(payload) !=3 so it will stop the execution of for-each record



- For Each Use Cases:

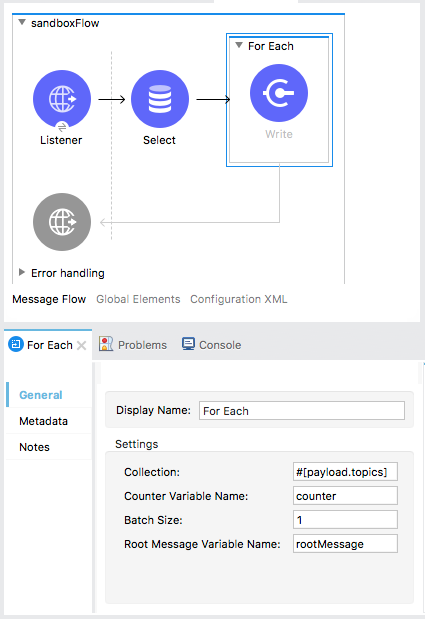
* Sequential Processing Required
* Synchronous Processing Required
* Small Data Set
* Processing of records in Batch required
* Process records only if previous records are processed successfully

-For each works in sequential manner that means if we have 100 records to process then it will take one-one record n process it unlike batch n parallel foreach where we can use parallel processing of the records.

-For-each scope does not modify the current payload.the output payload is same as the input payload what does that means is let say we have a flow where after listener we are setting the payload with set payload component and after the set payload we have for-each scope where we are setting the payload with set payload component n after the for-each scope we have logger which prints the payload so in logger we will get the paylaod that we set at starting of the flow just after the listener and we wont get the payload that we set in for-each scope.

-For non-java collections such as XML,json,use dataweave expression to split the data .use the collection field for this purpose.

* -In the following example, the **Collection** field in For Each is set to iterate over an array stored in payload.topics:



The For Each scope stores each item of the collection in payload during each iteration.

You can also split an array into batches to enable quicker processing. Each batch is treated as a separate Mule message. For example, if a collection has 200 elements and you set **Batch Size** to 50, the For Each scope iteratively processes 4 batches of 50 elements, each as a separate Mule message.

### Example XML

This is an example XML based on the For Each scope configuration detailed above:

...

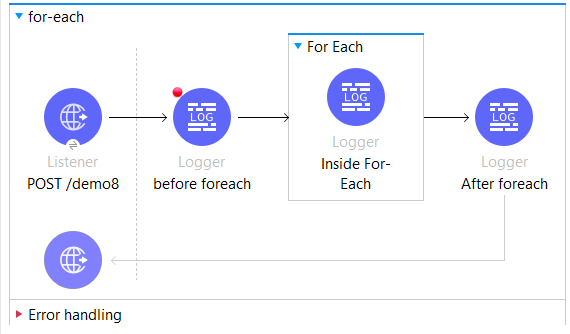
<foreach doc:name="For Each" collection="#[payload.topics]" batchSize="1" rootMessageVariableName="rootMessage" counterVariableName="counter">

<file:write ... >

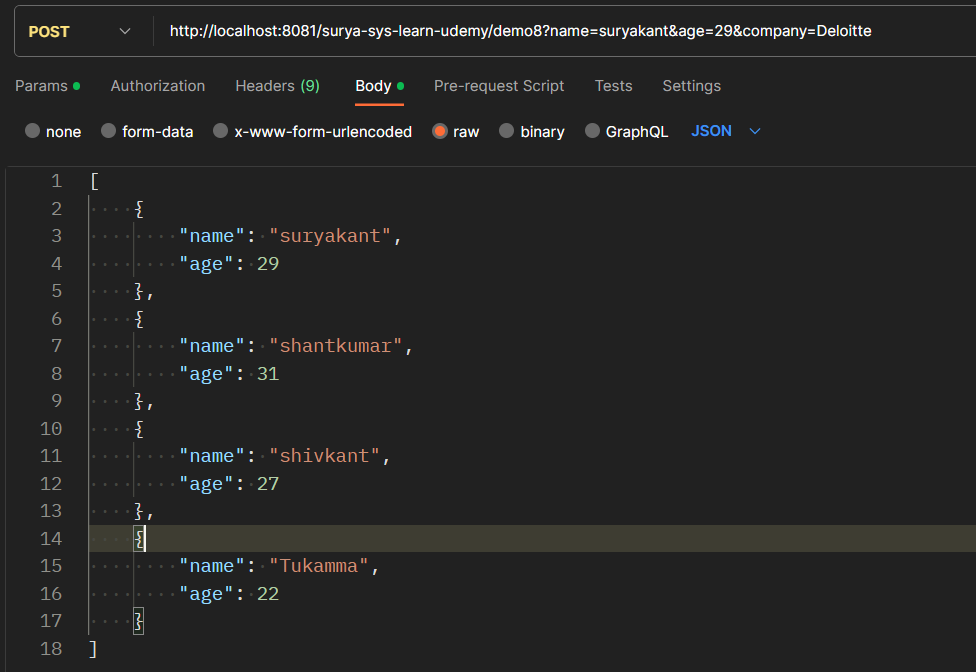
<!--Any other module that you want to include in the For Each scope -->

</foreach>

...







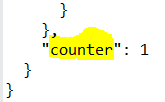
In this example we are passing payload as an Array of objects which contains name n age so we are passing 4 different objects in an array.

Note: When the trigger goes inside for-each scope the attributes become null n complete payload, attributes are stored in rootmessagevariablename variable. With rootmessagevariable name we have counter variable also defined at for-each to count the no of iterations. Variable which we define in our flow will be accessible inside for-each also.

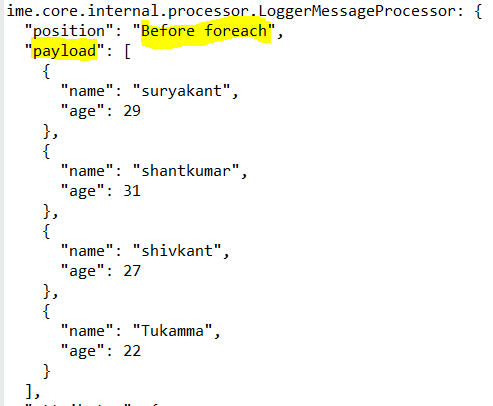


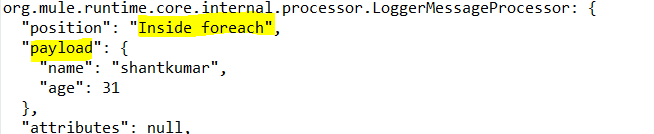
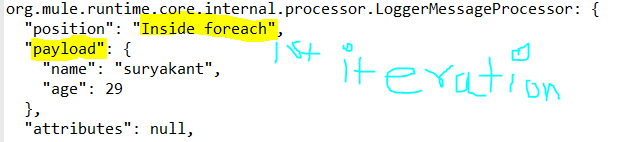


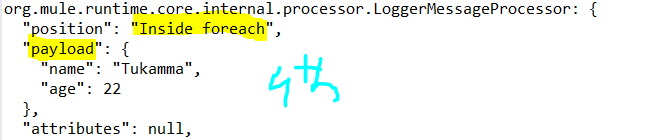
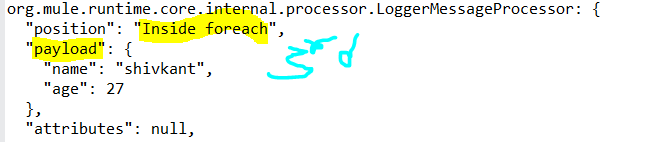




When batchsize=1 n defined payload has an array of 4 objects,so for-each will iterate over 4 times from each payload object n after foreach we will get the original payload which is same as before foreach payload

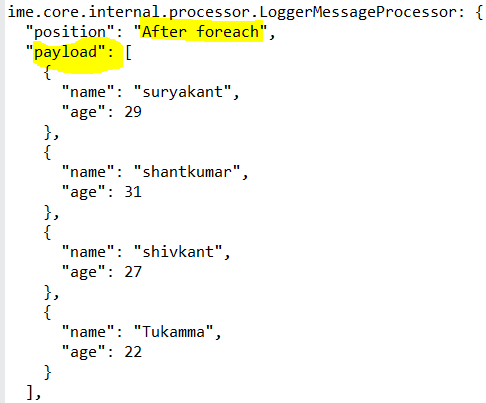
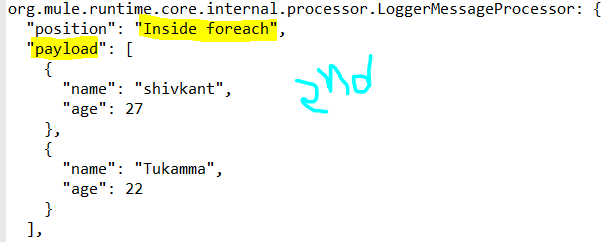
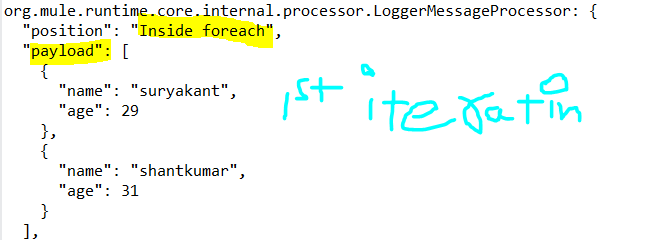
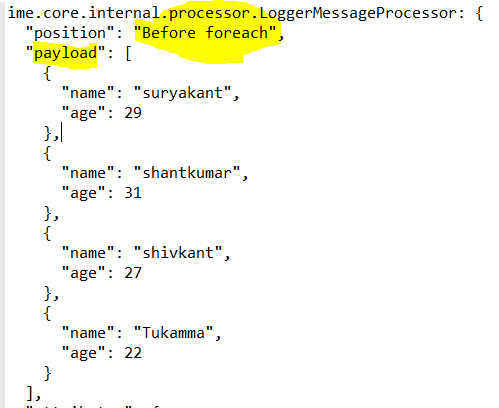




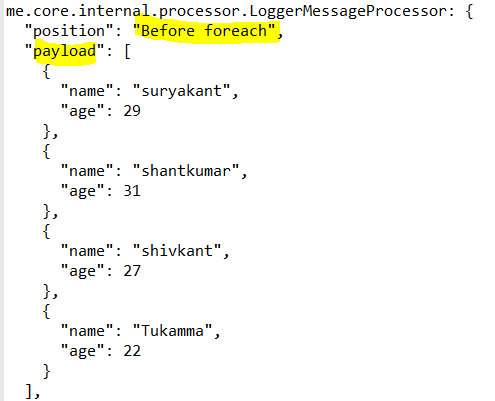


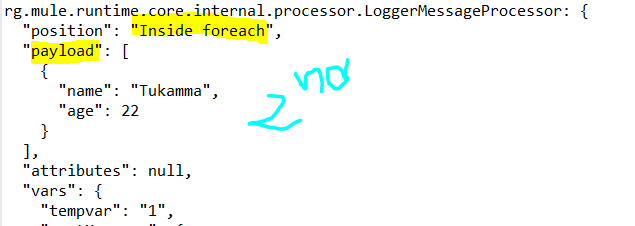
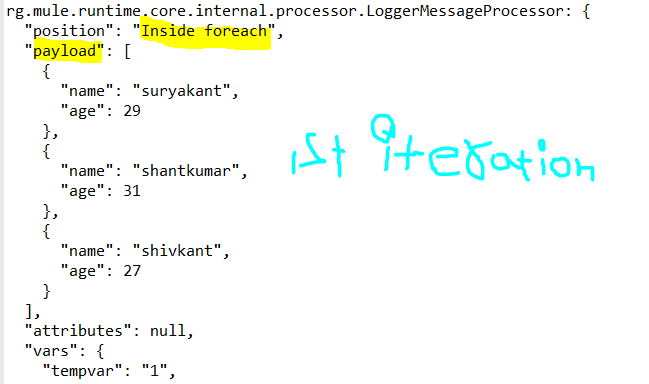


When batchsize=2, and we have payload with array of 4 objects ,it will take 2 objects in 1st iteration n remaining 2 object in 2nd iteration



When batchsize=3 and we have payload with array of 4 objects ,it will take 3 objects in 1st iteration n remaining 1 object in 2nd iteration





## Variable Propagation

Every execution of the For Each scope starts with the variables and values from the previous execution of the block. New variables or modifications to existing variables that take place when processing one element are visible during the processing of another element. These changes to variables continue to be available outside the For Each scope.

<set-variable variableName="var1" value="var1"/>

<set-variable variableName="var2" value="var2"/>

<foreach collection="#[['apple', 'banana', 'orange']]">

<choice>

<when expression="#[payload == 'apple']">

<set-variable variableName="var2" value="newValue"/>

<set-variable variableName="var3" value="appleVal"/>

</when>

<when expression="#[payload == 'banana']">

<set-variable variableName="var3" value="#[vars.var3 ++ ' bananaVal']"/>

<!-- var3 will now have value 'appleVal bananaVal'-->

</when>

<otherwise>

<set-variable variableName="var3" value="otherVal"/>

<set-variable variableName="var4" value="val4"/>

</otherwise>

</choice>

</foreach>

After aggregation, the variables are:

{var1: "var1", var2: "newValue", var3: "otherVal", var4: "val4"}

## Error Handling

If one of the elements in a collection throws an exception, the For Each scope stops processing that collection and invokes the error handler.

Parallel For Each(Synchronous Processing):

- The Parallel For Each scope splits the payload(collection) into parts that are simultaneously processed(parallely) in separate routes within the scope of any limitation configured for concurrent-processing. After all messages are processed, the results are aggregated following the same order they were in before the split, and then the flow continues.

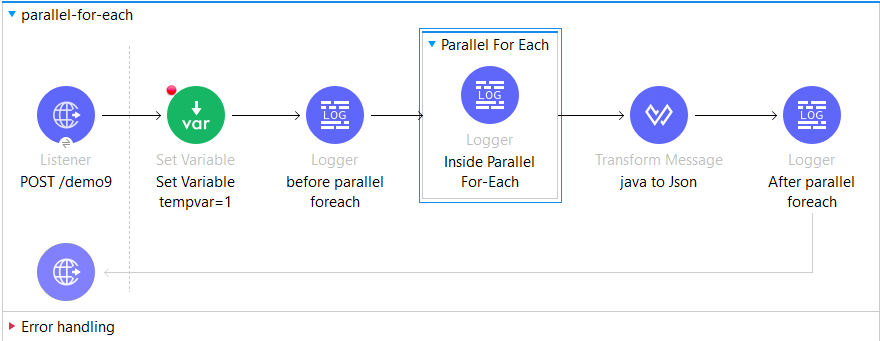
- Parallel For Each buffers all processing routes' results in a list to return it after the scope finishes processing, which can cause out-of-memory errors when processing a high number of entries. To process large payloads, use [Batch Processing](https://docs.mulesoft.com/mule-runtime/latest/batch-processing-concept) instead.

- Parallel for-each works in parallel synchronous manner

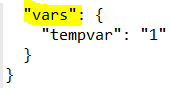
-Parallel For Each Use Cases:

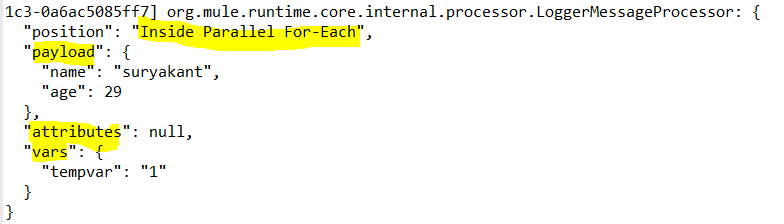
* Synchronous Processing Required with parallelism
* Suitable for Medium Data Set
* Provides Accumulated Output Required
* Process records irrespective of previous records status

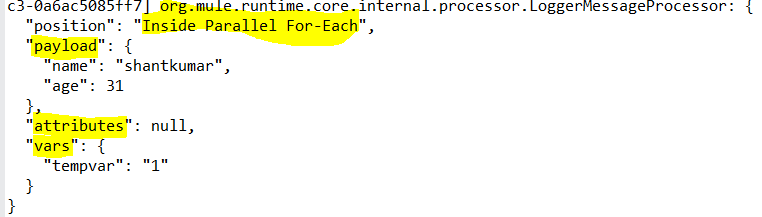
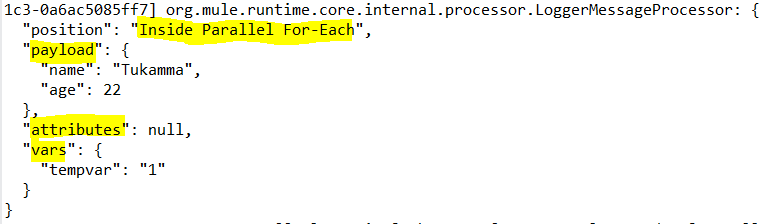
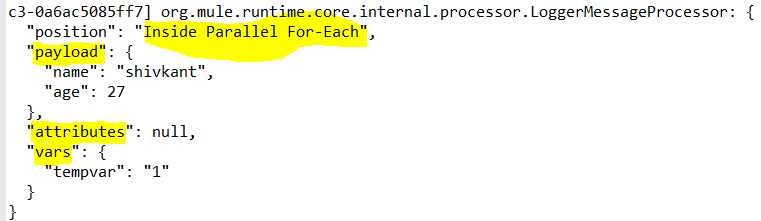
Note: Attributes inside parallel for-each will be ignored n kept as NULL

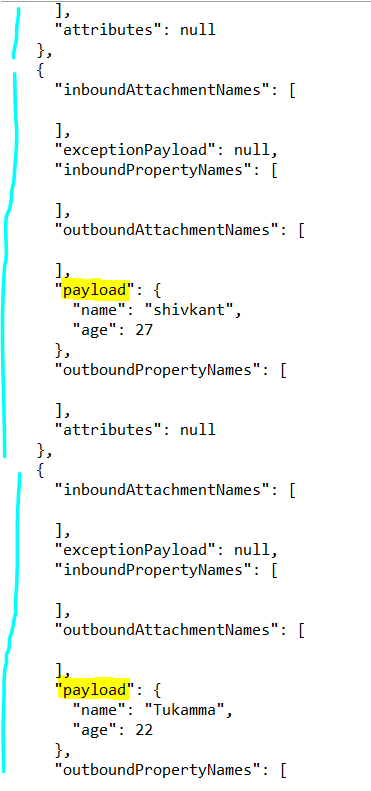


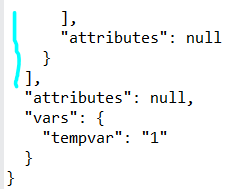






-By default if we don’t set target variable name inside parallel for-each scope then the payload contains array of objects of each route i.e part of each section of payload.

-if you see the above logs where we have Inside Parallel For-Each logger we can see that since it is parallel it accepts records in any order Suryakant,shantkumar,tukamma,shivkant

[

    {

        "inboundAttachmentNames": [],

        "exceptionPayload": null,

        "inboundPropertyNames": [],

        "outboundAttachmentNames": [],

        "payload": {

            "name": "suryakant",

            "age": 29

        },

        "outboundPropertyNames": [],

        "attributes": null

    },

    {

        "inboundAttachmentNames": [],

        "exceptionPayload": null,

        "inboundPropertyNames": [],

        "outboundAttachmentNames": [],

        "payload": {

            "name": "shantkumar",

            "age": 31

        },

        "outboundPropertyNames": [],

        "attributes": null

    },

    {

        "inboundAttachmentNames": [],

        "exceptionPayload": null,

        "inboundPropertyNames": [],

        "outboundAttachmentNames": [],

        "payload": {

            "name": "shivkant",

            "age": 27

        },

        "outboundPropertyNames": [],

        "attributes": null

    },

    {

        "inboundAttachmentNames": [],

        "exceptionPayload": null,

        "inboundPropertyNames": [],

        "outboundAttachmentNames": [],

        "payload": {

            "name": "Tukamma",

            "age": 22

        },

        "outboundPropertyNames": [],

        "attributes": null

    }

]

-if we set max-concurrency=1, then we get the payload processed as it is came in order like first it processes suryakant then shantkumar then shivkant then tukamma

- if we set max-concurrency=2, then we get the payload processed as it is came in order like first it processes suryakant then shantkumar then shivkant then tukamma

-Maxconcurrency allows to set the how many records should be processed at a time i.e sets value for parallelism.if we want to process sequential n one record at a time then we can set the max-concurrency=1

Note: maxConcurrency should not be 0 otherwise will give the error ‘maxConcurrency > 0 required but it was 0’

**-Error  Handling**

Because every route is processed in parallel, if an error is thrown in one route, processing continues in all of the other routes until all finish processing.

So, a failure router will not stop the processing of other routes which are executing in parallel.

We can get Success and Failure using below Dataweave.

It’s throwing Mule exception type **MULE:COMPOSITE\_ROUTING**

%dw 2.0

output application/json

---

{

"Success" : error.errorMessage.payload.results,

"Failures" : error.errorMessage.payload.failures

}

The above dw snippet gives payload as below in case of success and failure:

{

    "Success": {

        "1": {

            "inboundAttachmentNames": [],

            "exceptionPayload": null,

            "inboundPropertyNames": [],

            "outboundAttachmentNames": [],

            "outboundPropertyNames": [],

            "payload": {

                "name": "shantkumar",

                "age": 31

            },

            "attributes": null

        },

        "2": {

            "inboundAttachmentNames": [],

            "exceptionPayload": null,

            "inboundPropertyNames": [],

            "outboundAttachmentNames": [],

            "outboundPropertyNames": [],

            "payload": {

                "name": "shivkant",

                "age": 27

            },

            "attributes": null

        },

        "3": {

            "inboundAttachmentNames": [],

            "exceptionPayload": null,

            "inboundPropertyNames": [],

            "outboundAttachmentNames": [],

            "outboundPropertyNames": [],

            "payload": {

                "name": "Tukamma",

                "age": 22

            },

            "attributes": null

        }

    },

    "Failures": {

        "0": {

            "suppressedErrors": [],

            "errorType": {

                "identifier": "ANY",

                "parentErrorType": null,

                "namespace": "MULE"

            },

            "childErrors": [],

            "errorMessage": null,

            "cause": {

                "verboseMessage": "\r\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\nMessage               : payload.age is 29\r\nElement               : (None)\r\nElement DSL           : (None)\r\nError type            : (None)\r\nFlowStack             : (None)\r\n--------------------------------------------------------------------------------\r\nRoot Exception stack trace:\r\norg.mule.runtime.api.exception.DefaultMuleException: payload.age is 29\r\n\r\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\n",

                "localizedMessage": "payload.age is 29",

                "summaryMessage": "\r\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\nMessage               : payload.age is 29\r\nElement               : (None)\r\nElement DSL           : (None)\r\nError type            : (None)\r\nFlowStack             : (None)\r\n\r\n  (set debug level logging or '-Dmule.verbose.exceptions=true' for everything)\r\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\n",

                "cause": null,

                "message": "payload.age is 29",

                "i18nMessage": {

                    "code": -1,

                    "nextMessage": null,

                    "message": "payload.age is 29",

                    "args": []

                },

                "detailedMessage": "\r\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\nMessage               : payload.age is 29\r\nElement               : (None)\r\nElement DSL           : (None)\r\nError type            : (None)\r\nFlowStack             : (None)\r\n\r\n  (set debug level logging or '-Dmule.verbose.exceptions=true' for everything)\r\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\n",

                "additionalInfo": {},

                "messageCode": -1,

                "stackTrace": [],

                "suppressed": [],

                "exceptionInfo": {

                    "dslSource": null,

                    "alreadyLogged": false,

                    "errorType": null,

                    "flowStack": null,

                    "location": null,

                    "suppressedCauses": []

                },

                "info": {}

            },

            "description": "payload.age is 29",

            "detailedDescription": "payload.age is 29",

            "failingComponent": "parallel-for-each/processors/2/processors/1/route/0/processors/0 @ surya-sys-learn-udemy:surya-sys-learn-udemy.xml:276 (Raise error)"

        }

    }

}

## For Each vs Parallel For Each

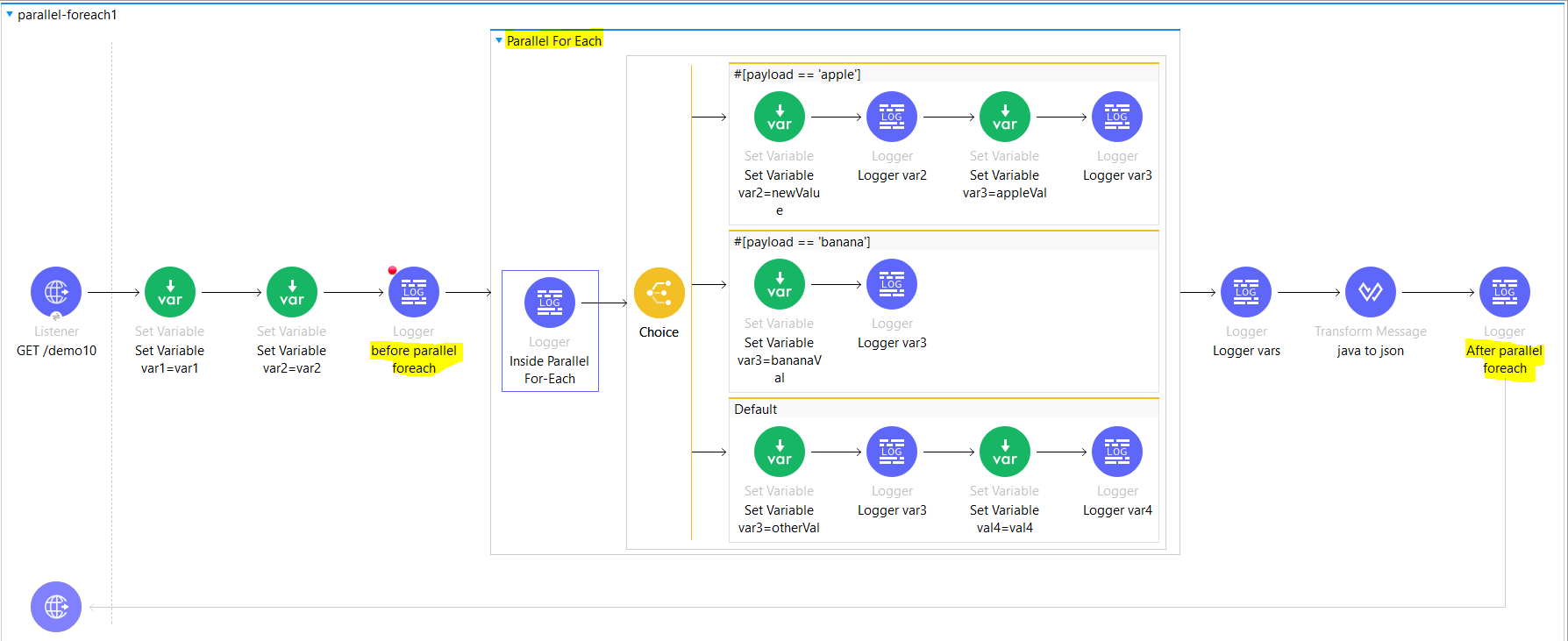
Both basically process the collection and each route runs with the same initial context.

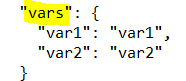
The difference between these two scopes are:

1. For Each works sequentially, while the Parallel For Each processes in parallel. This difference affects error handling. Because of the processing differences, the execution of For Each execution is interrupted when an error is raised (and the Error Handler is invoked), while Parallel For Each processes every route before invoking the Error Handler with a MULE:COMPOSITE\_ROUTE error type.
2. For Each does not modify the payload, while the Parallel For Each outputs a collection of the output messages from each iteration.

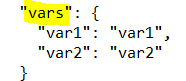
-Every execution of the Parallel For Each scope starts with the same variables and values as before the execution of the block.

New variables or modifications of already existing variables while processing one element are not visible while processing another element. All of those variable changes are not available outside the Parallel For Each scope, the set of variables (and their values) after the execution of the Parallel For Each Scope remains the same as before the execution.







None of the modifications done inside the Parallel For Each scope are registered, including the creation of new variables.

Batch processing (Asynchronous Parallel Processing):

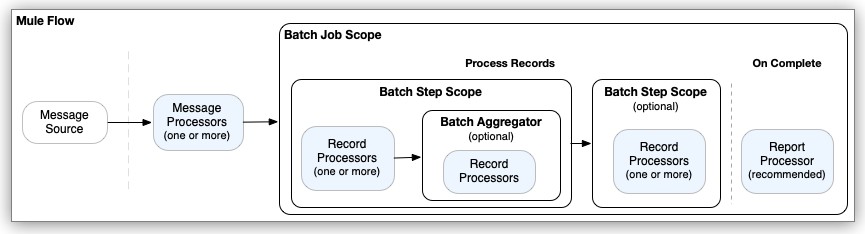
- Mule batch processing components are designed for reliable, asynchronous processing of larger-than-memory data sets.

-The components are the Batch Job, Batch Step, and Batch Aggregator. The Batch Job component automatically splits source data and stores it into persistent queues, which makes it possible to process large data sets while providing reliability. In the event that the application is redeployed or Mule crashes, the job execution is able to resume at the point it stopped.

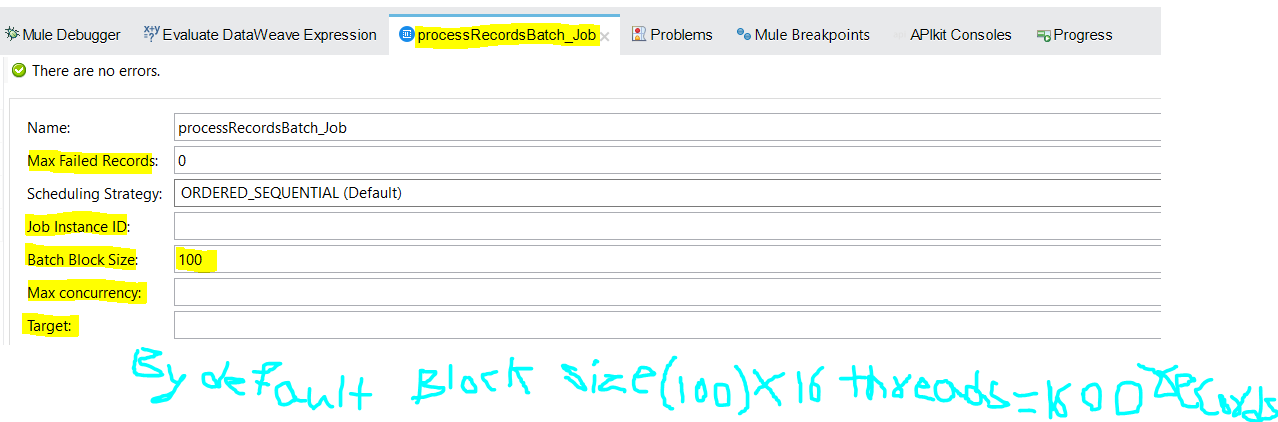
- Common use cases for batch processing include:

* Synchronizing data sets between business applications, such as syncing contacts between NetSuite and Salesforce.
* Extracting, transforming and loading (ETL) information into a target system, such as uploading data from a flat file (CSV) to Hadoop.

- Mule batch processing components prepare records for processing in batches, run processors on those records, and issue a report on the results of the processing. Record preparation and reporting take place within the Batch Job component. Processing takes place within one or more Batch Step components and, optionally, a Batch Aggregator component within a Batch Step component.

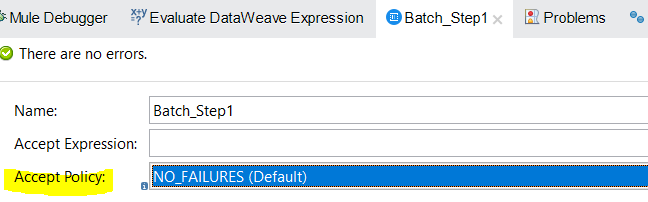


The processors you place within batch processing components act on records. Each record is similar to a Mule event: Processors can access, modify, and route each record payload using the payload keyword, and they can work on Mule variables using vars. However, they cannot access or modify Mule attributes (attributes) from the input to the Batch Job component.



Let say we have 20000 records which needs to be sync with salesforce so we can use batch job to create batch of records and processed in async manner where By default batch block size is (100) n batch job can run 16 threads= 100\*16 =1600 records can be in one batch so totally will have 13 batches of records to process all the 20k records…1600\*13(no of batches)= 20k+,if we want then we can increase the batch size by any number so let say we increased batch block size to 400 then the 400\*16=6400 records in a batch so we need 4 batches of records to process all the 20k+ records 6400\*4=25k+

Maxconcurrency is used to set max level of parallelism i.e at a time maximum how many records can get processed if we set 1 then it will be sequential n at a time only 1 record get processed.if no value set then it will be by default it will be 2\* no of available cores in the cpu



Batch step configuration has accept policy which can specify only failures or no failures or all types of records should get accepted at batch step.Mule retains a list of all records that succeed or fail to process through each Batch Step. If an event processor in a Batch Step fails to process a record, Mule continues processing the batch, skipping over the failed record in each subsequent Batch Step based on the acceptPolicy. The Batch Job component provides the maxFailedRecords property for setting the number records that can fail before the batch job stops.

-Batch Job scope has two sections:

* Process Records(used to process the records by using Batch-Step component,which has two sections in it processors(components) and aggregator(optional),to use aggregator we have to drag n drop batch-aggregator component)
* On-Complete (it is used to generate the report of records processed like how many got successful n how many got error.

# Batch Job Phases

Each time a Batch Job component executes, the following phases take place:

1. [Load and Dispatch Phase](https://docs.mulesoft.com/mule-runtime/latest/batch-phases#phase_load_dispatch): The Batch Job component splits [valid input](https://docs.mulesoft.com/mule-runtime/latest/batch-processing-concept#valid_input) into records and prepares the records for processing. This phase takes place within the Batch Job component. Within the scope of the Batch Job component, Mule exposes the currently processing batch job instance ID through the batchJobInstanceId variable. You can use vars.batchJobInstanceId to access the identifier of the current instance in any batch processing phase. The Batch Job component automatically splits the received message payload into records and stores the records in the stepping queue.The Batch Job component either successfully generates and queues a record for every item from the input, or the entire event fails with an error type, such as MULE:EXPRESSION message, and an error message, such as "Expecting Array or Object but got String." evaluating expression: "payload".
2. [Process Phase](https://docs.mulesoft.com/mule-runtime/latest/batch-phases#phase_process): Mule components and connector operations within one or more Batch Step components in the Batch Job process records within a given batch job instance. Processing within a Batch Aggregator component also occurs in the Process phase.
3. [On Complete Phase](https://docs.mulesoft.com/mule-runtime/latest/batch-phases#phase_on_complete): The Batch Job component issues a report object with the result of processing the batch job instance. During this phase, you can configure Mule runtime to create a report or summary of the records it processed in a given batch job instance. This phase provides system administrators and developers insight into which records failed or succeeded, but it does not process or provide access to individual records, nor does it pass processed records to downstream processors in the flow.

At the end of this phase, the batch job instance completes and ceases to exist.

As a best practice, configure a mechanism for reporting on failed or successful records to facilitate further action where required. During the On Complete phase, you can perform either of these tasks:

* Reference the result object of the batch job instance from elsewhere in the Mule application to capture and use batch job metadata, such as the number of records that failed to process in a particular batch job instance.
* Log the result object for each batch job instance:

<batch:job name="Batch3">

<batch:process-records>

<batch:step name="Step1">

<batch:record-variable-transformer/>

<ee:transform/>

</batch:step>

<batch:step name="Step2">

<logger/>

<http:request/>

</batch:step>

</batch:process-records>

<batch:on-complete>

<logger level="INFO" doc:name="Logger"

message='#[payload as Object]'/>

</batch:on-complete>

</batch:job>

From the logger set to payload as Object, a report looks something like this:

INFO 2022-07-06 11:39:02,921 [[MuleRuntime].uber.06:

[w-batch-take6].batch-management-work-manager @56978b97]

[processor: w-batch-take6Flow/processors/3/route/1/processors/0;

event: e835b2c0-fd5a-11ec-84a5-147ddaaf4f97]

org.mule.runtime.core.internal.processor.LoggerMessageProcessor:

{onCompletePhaseException=null, loadingPhaseException=null, totalRecords=1000,

elapsedTimeInMillis=117, failedOnCompletePhase=false, failedRecords=0,

loadedRecords=1000, failedOnInputPhase=false, successfulRecords=1000,

inputPhaseException=null, processedRecords=10, failedOnLoadingPhase=false,

batchJobInstanceId=e84b5da0-fd5a-11ec-84a5-147ddaaf4f97}

The fields in the batch job report object are accessible as keys when using DataWeave selectors, such as payload.failedRecords to return the number of failed records in the instance.

If you leave the On Complete phase empty, the batch job instance silently completes, and the logs provide processing information about the instance, for example:

Finished execution for instance 'e84b5da0-fd5a-11ec-84a5-147ddaaf4f97'

of job 'w-batch-take6Batch\_Job'.

Total Records processed: 1000. Successful records: 1000. Failed Records: 0

## Filtering Records to Process within a Batch Step Component

You can apply one or more record filters as attributes to any number of Batch Step components. For example, assume that the first Batch Step component within a given Batch Job component checks for a Salesforce contact in each record and the second updates the contact information in those records. To ensure that the second Batch Step processes only records that succeeded in the first step, you can configure a filter on the second Batch Step. The filter streamlines processing so that Mule focuses on only the relevant data for a given step.

To filter records, the Batch Step component supports one acceptExpression and one acceptPolicy. Both are optional, and no other Mule components accept these filters. If you use both filters on the same component, Mule evaluates the filters in the following order:

1. [acceptExpression](https://docs.mulesoft.com/mule-runtime/latest/batch-filters-and-batch-aggregator#accept_expression)
2. [acceptPolicy](https://docs.mulesoft.com/mule-runtime/latest/batch-filters-and-batch-aggregator#accept_policy)

A Batch Step component that uses an acceptExpression attribute applies a DataWeave expression to each record that reaches the component and accepts a record for processing within the component only if the expression evaluates to true for that record. If the record evaluates to false, the Batch Step component skips the record, and that record becomes available to the next Batch Step component within the Batch Job component if one exists.

The example below filters out all records in which the age value is less than 21:

<batch:job jobName="batchJob">

<batch:process-records >

<batch:step name="adultsOnlyStep" acceptExpression="#[payload.age > 21]">

...

</batch:step>

</batch:process-records>

</batch:job>

A Batch Step component uses an acceptPolicy attribute to apply a policy to records processed in a previous Batch Step component. The Batch Step accepts a record for processing within the component only if the policy evaluates to true for that record. The default policy is NO\_FAILURES. Other policies are ONLY\_FAILURES and ALL.

In the following example, the second Batch Step component (batchStep2) accepts only those records that failed to process successfully during the preceding step (batchStep1). For example, assume processors within batchStep1 check each record for a Salesforce contact. If the check fails to find a contact, batchStep2 accepts only those records without contacts, skipping records with contacts. Message processors within batchStep2 can add a contact to those records. <batch:job jobName="batchJob">

<batch:process-records >

<batch:step name="batchStep1" >

<!-- Check for contact -->

...

</batch:step>

<batch:step name="batchStep2" accept-policy="ONLY\_FAILURES">

<!-- Accept records that failed -->

...

</batch:step>

</batch:process-records>

</batch:job>

For demonstration purposes, the following example uses the Raise Error component (<raise-error />) to produce failures in the first Batch Step (batchStep1). The second Batch Step (batchStep2) accepts failed records only (ONLY\_FAILURES) and logs the payload of those records. The third uses the default acceptPolicy to accept NO\_FAILURES.

<flow name="batch-exampleFlow" >

<scheduler doc:name="Scheduler" >

<scheduling-strategy >

<fixed-frequency frequency="75" timeUnit="SECONDS"/>

</scheduling-strategy>

</scheduler>

<!-- Set Payload -->

<set-payload value="#[[1,2,3,4,5]]" doc:name="Set Payload" />

<!-- Batch Job Component -->

<batch:job jobName="Batch\_Job" maxFailedRecords="-1">

<batch:process-records >

<!-- First Batch Step defaults to NO\_FAILURES -->

<batch:step name="batchStep1" acceptPolicy="ALL"

acceptExpression="#[payload &lt; 3]">

<!-- Raising Error -->

<raise-error doc:name="Raise error"

type="MY:APP" description="Example: Raising Error "/>

</batch:step>

<!-- Second Batch Step: ONLY FAILURES -->

<batch:step name="batchStep2" acceptPolicy="ONLY\_FAILURES">

<logger level="INFO" doc:name="Logger" message="#[payload]"

category="LOGGER IN SECOND BATCH STEP"/>

</batch:step>

<batch:step name="batchStep3" >

<logger level="INFO" doc:name="Logger" message="#[payload]"

category="LOGS IN THIRD BATCH STEP"/>

</batch:step>

</batch:process-records>

</batch:job>

</flow>

* The Set Payload component (<set-payload />) sends the array [1,2,3,4,5] to the Batch Job component.
* The Batch Job component (<batch:job />) sets maxFailedRecords to -1 so that failed records within the first Batch Step component do not stop the batch job instance from processing.
* The first Batch Step (batchStep1) configures acceptExpression="#[payload < 3]" to accept only records with a payload less than 3 and throws an error on those records because of the Raise Error component within it:

INFO ...DefaultBatchStep: Found exception processing record on step 'batchStep1' for job instance 'f9bb6d60-5ef2-11ed-81e7-147ddaaf4f97'

of job 'batch-example-policiesBatch\_Job'.

The second Batch Step (batchStep2) prints the payload of the failed records, for example:

INFO ... LOGGER IN SECOND BATCH STEP: 1

INFO ... LOGGER IN SECOND BATCH STEP: 2

The third Batch Step (batchStep3) prints the payload of the successful records, for example:

INFO ... LOGS IN THIRD BATCH STEP: 3

INFO ...LOGS IN THIRD BATCH STEP: 4

INFO ...LOGS IN THIRD BATCH STEP: 5

After the batch job instance completes, the logs indicate that 3 records processed successfully and 2 records failed:

INFO ...engine.DefaultBatchEngine: Finished execution for instance ...

Total Records processed: 5. Successful records: 3. Failed Records: 2

## Performing Bulk Operations from a Batch Aggregator Component

Aggregation is useful for sending multiple records within an array to an external server. Within the Batch Aggregator component, you can add an operation, such as a bulk upsert, insert, or update operation, to load multiple records to a server with a single execution of an operation, instead of running an operation separately on each record.

You can process records in separate arrays of a fixed size or stream a single array of records from the batch job instance. For example, you can configure an Upsert operation in the Anypoint Connector for Salesforce (Salesforce Connector) to upsert 200 processed records. Alternatively, you can stream all the records in the instance to the server.

The following example bulk upserts records in separate arrays of 200 records per upsert:

<batch:job jobName="batchJob">

<batch:process-records >

<batch:step name="batchStep">

<batch:aggregator size="200">

<salesforce:upsert doc:name="Upsert" ... />

</batch:aggregator>

</batch:step>

</batch:process-records>

</batch:job>

The following example streams upserts to a database:

<batch:job jobName="batchJob">

<batch:process-records >

<batch:step name="batchStep">

<batch:aggregator streaming="true">

salesforce:upsert doc:name="Upsert" ... />

</batch:aggregator>

</batch:step>

</batch:process-records>

</batch:job>

Error handling:  
Some connectors handle record-level errors without causing an entire batch aggregation process to fail, for example, [Anypoint Connector for Salesforce (Salesforce Connector)](https://docs.mulesoft.com/salesforce-connector/10.18/salesforce-connector-reference" \l "upsert) and [Anypoint Connector for NetSuite (NetSuite Connector)](https://docs.mulesoft.com/netsuite-connector/11.7/netsuite-reference" \l "Upsert). At runtime, these connectors keep track of records that the target resource accepts successfully and which fail. Rather than failing the entire group of records, the connector upserts as many records as it can and tracks any failures for notification purposes.

## Modifying Records within a Batch Aggregator

You can modify records within the Batch Aggregator component, just as you can modify them with processors in the Batch Step component. The modifications can take place sequentially or through random access to specific records.

### Sequential Processing within a Batch Aggregator

The following example performs sequential access to records. Using Groovy within a [Scripting module](https://docs.mulesoft.com/scripting-module/2.0/), the example modifies the payload of each record from the [For Each](https://docs.mulesoft.com/mule-runtime/latest/for-each-scope-concept) component’s iteration and creates a variable for each collected record.

<batch:job jobName="batchJob">

<batch:process-records>

<batch:step name="batchStep">

<batch:aggregator doc:name="batchAggregator" size="10">

<foreach doc:name="For Each">

<script:execute engine="groovy">

<script:code>

vars['marco'] = 'polo'

vars['record'].payload = 'hello'

</script:code>

</script:execute>

</foreach>

</batch:aggregator>

</batch:step>

</batch:process-records>

</batch:job>

### Randomly Accessing Records within a Batch Aggregator

You can also use the For Each scope when randomly accessing records by their iteration number. For Each exposes a records variable, which is an immutable list that it uses to keep track of the iteration. You can use this variable to randomly access records in the list from the Batch Aggregator component.

To demonstrate random access when using fixed-size aggregation, the following example specifies the index of a record in the For Each list of records. Instead of sequentially accessing each record, using the records variable with an index of a record selects a single record from the list. The example uses the Scripting module to modify the payload of the selected record and create a variable for that record:

<batch:job jobName="batchJob">

<batch:process-records>

<batch:step name="batchStep">

<batch:aggregator doc:name="batchAggregator" size="10">

<foreach doc:name="For Each">

<script:execute engine="groovy">

<script:code>

records[0].vars['marco'] = 'polo'

records[0].vars['record'].payload = 'hello'

</script:code>

</script:execute>

</foreach>

</batch:aggregator>

</batch:step>

</batch:process-records>

</batch:job

You can configure a Batch Aggregator component to stream its content. Setting this component to stream the records (streaming="true") enables you to process an array of all the records in the batch job instance without running out of memory, regardless of how many or how large the records are. For example, if you need to write millions of records to a CSV file, you can stream the records with the Batch Aggregator component.

<batch:job jobName="batchJob">

<batch:process-records >

<batch:step name="batchStep">

<batch:aggregator streaming="true">

<file:write path="reallyLarge.csv">

<file:content><![CDATA[%dw 2.0

...

}]]></file:content>

</batch:aggregator>

</batch:step>

</batch:process-records>

</batch:job>

## Preserving the MIME types of the Aggregated Records

Aggregated records pass into the Batch Aggregator component as an array containing each record’s payload. However, by default, the MIME types associated with those payloads are not preserved. To preserve record’s MIME types, specify the preserveMimeTypes attribute in the Batch Aggregator component.

Consider the following JSON array:

[

{

"name": "Tony Stark",

"alias": "Iron Man",

"status": "DEAD"

},

{

"name": "Steve Rodgers",

"alias": "Captain America",

"status": "RETIRED"

},

{

"name": "Peter Parker",

"alias": "SpiderMan",

"status": "FUGITIVE"

}

] Assume that the [JSON array](https://docs.mulesoft.com/mule-runtime/latest/batch-filters-and-batch-aggregator#json_array) is the input to the following Batch Job component:

<batch:job name="avengersLogger">

<batch:process-records>

<batch:step name="log">

<batch:aggregator size="10">

<foreach>

<logger message="Agent #[payload.alias] is #[payload.status]" />

</foreach>

</batch:aggregator>

</batch:step>

</batch:process-records>

</batch:job>

The batch engine splits the input JSON array into individual records, which aggregator block receives an array with three elements. The first one of them is:

{

"name": "Tony Stark",

"alias": "Iron Man",

"status": "DEAD"

}

However, when the Logger component attempts to evaluate the #[payload.alias] expression, an error similar to the following one results:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Message : "You called the function 'Value Selector' with these arguments:

1: Binary ("ewogICJmaXJzdE5hbWUiOiAiUmFtIiwKICAibGFzdE5hbWUiOiAiUmFtMSIsCiAgImFkZHJlc3Mi...)

2: Name ("alias")

But it expects one of these combinations:

(Array, Name)

(Array, String)

(Date, Name)

(DateTime, Name)

(LocalDateTime, Name)

(LocalTime, Name)

(Object, Name)

(Object, String)

(Period, Name)

(Time, Name)

5| name: payload.alias,

^^^^^^^^^^^^^

This error occurs because MIME types are not preserved, which prevents Mule from reading it as JSON. To resolve this issue, you must set the preserveMimeTypes attribute in the Batch Aggregator component to true, for example:

<batch:aggregator size="10" preserveMimeTypes="true">

<foreach>

<logger message="Agent #[payload.alias] is #[payload.status]" />

</foreach>

</batch:aggregator>

The setting makes Mule maintain each record’s media type and treat the payload as a JSON document.

## Changing the Record Block Size

A traditional online processing model typically maps each request to a worker thread. Generally, there is 1:1 relationship between a request and a running thread regardless of the processing type. This relationship holds for synchronous, asynchronous, one-way, and request-response processing types, and when temporarily buffering requests before processing.

However, in a batch job, the traditional threading model does not apply because records are first stored in a persistent queue, before the Process phase begins.

To improve performance, Mule runtime queues and schedules batch records in blocks of up to 100 records per thread. This behavior reduces the number of I/O requests and improves an operation’s load. Batch jobs use Mule thread pools, so there is no default for the job. Each thread iterates through that block to process each record, and then each block is queued back, and the process continues.

Consider having 1 million records to place in a queue for a 3-step batch job. At least three million I/O operations occur as the Mule runtime engine takes and requests each record as they move through the job’s phases.  
Performance requires having enough available memory to process the threads in parallel, which means moving the records from persistent storage into RAM. The larger your records and their quantity, the more available memory you need for batch processing.

Although the standard model of up to 100 records per thread in the batch job works for most use cases, consider three use cases where you need to increase or decrease the block size:

* Assume you have 200 records to process through a batch job. With the default 100-record block size, Mule can only process two records in parallel at a time. If a batch job has fewer than 101 records to process, then processing becomes sequential. If you need to process heavy payloads, then queueing a hundred records demands a large amount of working memory.
* Consider a batch job that needs to process images, and an average image size of 3 MB. In this case, Mule processes 100-record blocks with payloads of 3 MB in each thread. Hence, your default threading-profile setting would require a large amount of working memory just to keep the blocks in the queue. In this case, set a lower block size to distribute each payload through more jobs and lessen the load on your available memory.
* Suppose you have 5 million records with payloads so small that you can fit blocks of 500 records in your memory without problems. Setting a larger block size improves your batch job time without sacrificing working memory load.

To take full advantage of this feature, you must understand how the block sizes affect your batch job. Running comparative tests with different values and testing performance helps you find an optimum block size before moving this change into production.

Remember that modifying the batch block size is optional. If you apply no changes, the default value is 100 records per block.

You set the size through the Batch Job component, for example:

<batch:job jobName="atch\_Job" blockSize="100">

...

</batch:job>

## Setting a Max Concurrency Limit on Batch Job Instances

The Max Concurrency (maxConcurrency) property limits the number of record blocks to process concurrently.

You can configure the maxConcurrency property as in the following example:

<batch:job jobName="test-batch" maxConcurrency="${batch.max.concurrency}">

...

</batch:job>

By default, the Batch Job component limits the maximum concurrency to twice the number of available cores. The capacity of the system running the Mule instance also limits concurrency.

# Handling Errors During Batch Job

Mule batch processing is designed to handle very large data sets and to perform almost real-time data integration that recovers from crashes and continues processing a job from a point of failure.

## Batch Processing Strategies for Error Handling

Mule has three options for handling a record-level error:

1. **Finish processing** Stop the execution of the current job instance. Finish the execution of the records currently in-flight, but do not pull any more records from the queues and set the job instance into a FAILURE state. The On Complete phase is invoked.
2. **Continue processing** the batch regardless of any failed records, using the acceptExpression and acceptPolicy attributes to instruct subsequent batch steps how to handle failed records.
3. **Continue processing** the batch regardless of any failed records (using the acceptExpression and acceptPolicy attributes to instruct subsequent batch steps how to handle failed records), until the batch job accumulates a **maximum number of failed records** at which point the execution will halt just like in option 1.

By default, Mule’s batch jobs follow the first error handling strategy which halts the batch instance execution. The above behavior is controlled through the maxFailedRecords attributes.

| **Failed Record Handling Option** | **Batch Job** | |
| --- | --- | --- |
|  | **Attribute** | **Value** |
| Stop processing when a failed record is found. | maxFailedRecords | 0 |
| Continue processing indefinitely, regardless of the number of failed records. | maxFailedRecords | -1 |
| Continue processing until reaching maximum number of failed records. | maxFailedRecords | integer |

<batch:job jobName="Batch1" maxFailedRecords="0">

### Crossing the Max Failed Threshold

When a batch job accumulates enough failed records to cross the maxFailedRecords threshold, Mule aborts processing for any remaining batch steps, skipping directly to the On Complete phase.

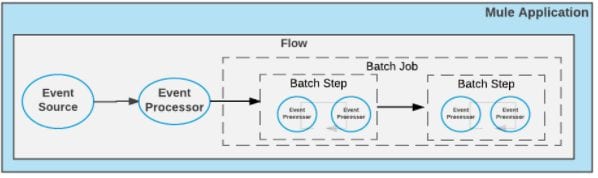
For example, if you set the value of maxFailedRecords to "10" and a batch job accumulates ten failed records in the first of three batch steps, Mule does not attempt to process the batch through the remaining two batch steps. Instead, it aborts further processing and skips directly to On Complete to report on the batch job failure.

If a batch job does not accumulate enough failed records to cross the maxFailedRecords threshold, all records – successes and failures – continue to flow from batch step to batch step; use filters to control which records each batch step processes.

Note: the components

1)batch job(if both sections like process records n on complete are empty then it will fail to deploy )

2)section like process records(atleast should have a 1 batch step which has atleast 1 processor in it and on Complete should have atleast 1 processor in it to successfully get deployed) .



-if any error occurs then The Behaviour can be configured.

-provides output as Original Payload same as the input.

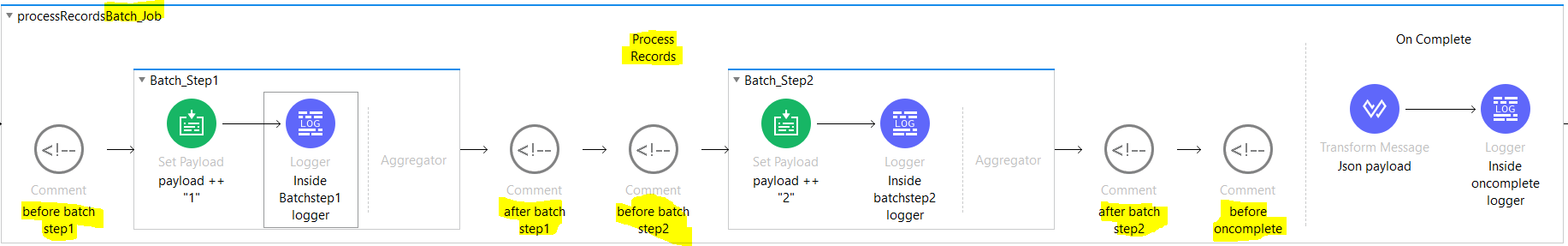
-Parallel For Batch Job Cases:

* Asynchronous Processing Required
* Ordering of process records not needed
* Large Data set
* Processing logic is complex and filtering is optional
* Process records irrespective of previous records status

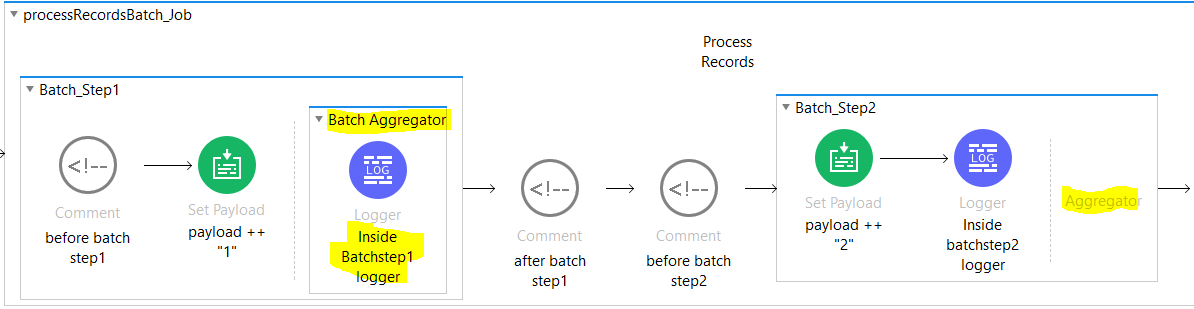
-In Batch job scope ,if we define any components outside or in-between the batch-step , on-complete phase within a batch job scope then we wont be able to deploy the project successfully will get error like Caused by: org.mule.runtime.api.exception.MuleRuntimeException: There was '1' error while parsing the given file 'surya-sys-learn-udemy.xml'.

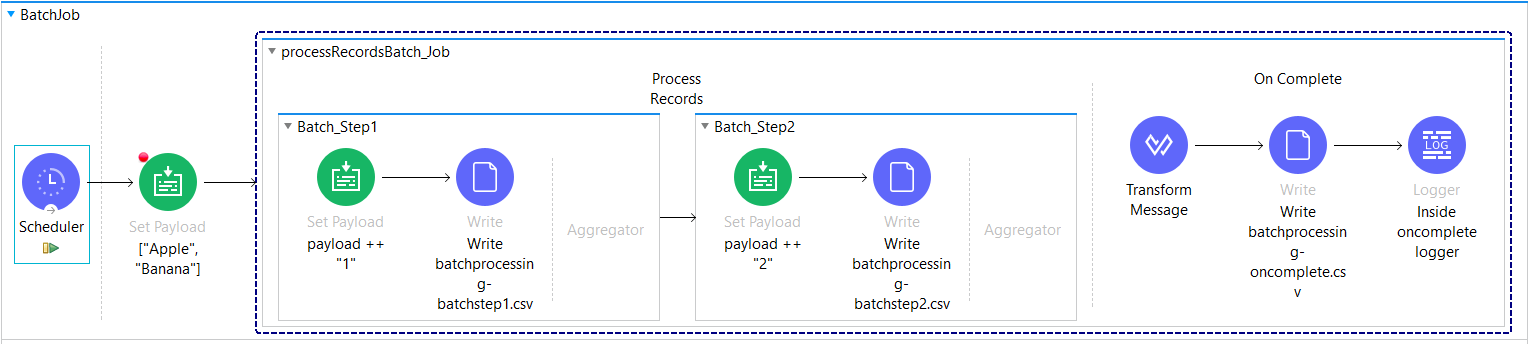
Full list:

org.xml.sax.SAXParseException; lineNumber: 413; columnNumber: 270; cvc-complex-type.2.4.a: Invalid content was found starting with element '{"http://www.mulesoft.org/schema/mule/core":logger}' (highlighted n also commented components which gives the error in below screenshot.)



* Plus in the above screenshot u can see tht we don’t have component defined in aggregator section of batch step,if we drag n drop any component aggregatoe wont allow it to add in aggregator section to add components we have to first place the batch aggregator in aggregator section n then we can add any components. Like below screenshot

- The following example outlines the XML structure of a Mule flow that performs batch processing and returns a report with the results of that processing:



In the above screenshot we are using ,

1)scheduler component which acts as a source n triggers the fbatchJob flow after every specified scheduling strategy.

2)We are using set payload which sets payload as an array of two strings [“Apple”,”Banana”]

3)We are declaring batch job component with all default values(max failed records=0,batch block size=100,scheduling strategy=ordered\_sequential) which means it has 100\*16=1600 batch of one block n as per out payload we have an array with only two records.batch job has two sections process records where we have two batch-steps n on complete.Batch job is used to split the source data in batch of records to easily get processed.it is used to prepare the records that means in our case we have payload=[“Apple”,”Banana”] so it will split n make it two records that will get processed

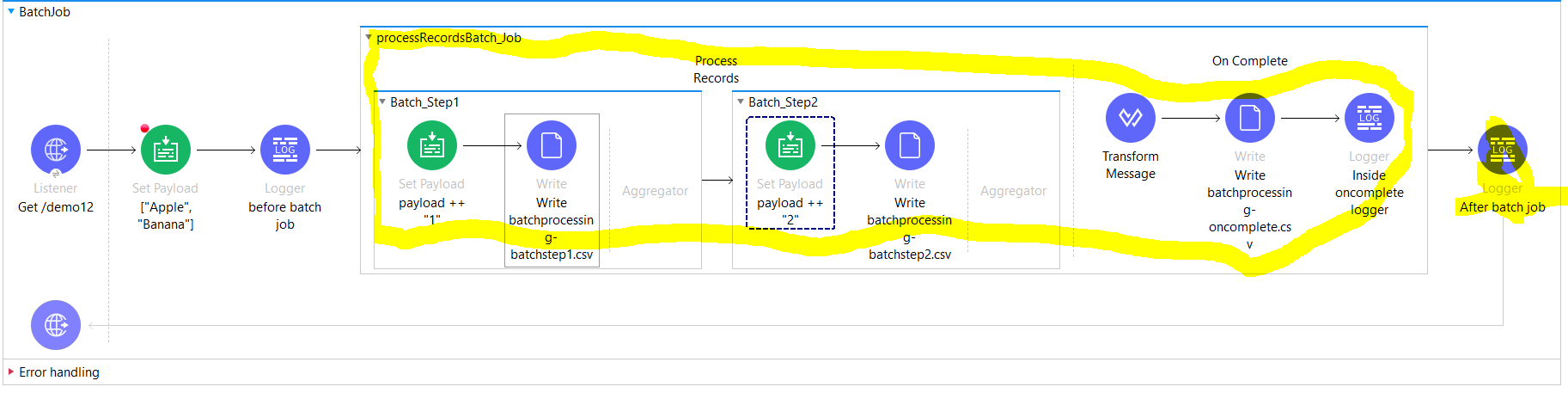
4) in process records we have 2 batch steps so it will take one-one records n process it in batch step1 once all records get processed in batch step1 then it will move to batch step2

In our batch step1 we are setting the payload with current record+1( let say first records is Apple so it will be Apple1 n then we are writing that record Apple1 in file once writing completes it will take 2nd record i.e Banana n will set the payload as Banana1 n then writes in the file since we have only two records now so the trigger moves to batch step2 where we again setting the current record+2 so it will take the current recors as return by the batch step 1 n will set the payload so let say it took Apple1 as record so it will set the payload as Apple12 n then writes it in the 2nd file then again goes and takes the second value i.e Banana1 n set the payload to Banana12 n writes it in a file once process records gets completes then on complete will get called

5)in on-complete section provides reporting of the records.where we have transform msg component which converts payload to json n the payload is nothing but the summary of records like total records ,failed records,successful records then we hve 3rd file where we are writing the summary of records n then we have logger which prints the summary in console.

-How do I make batch processing synchronous in mule 4?

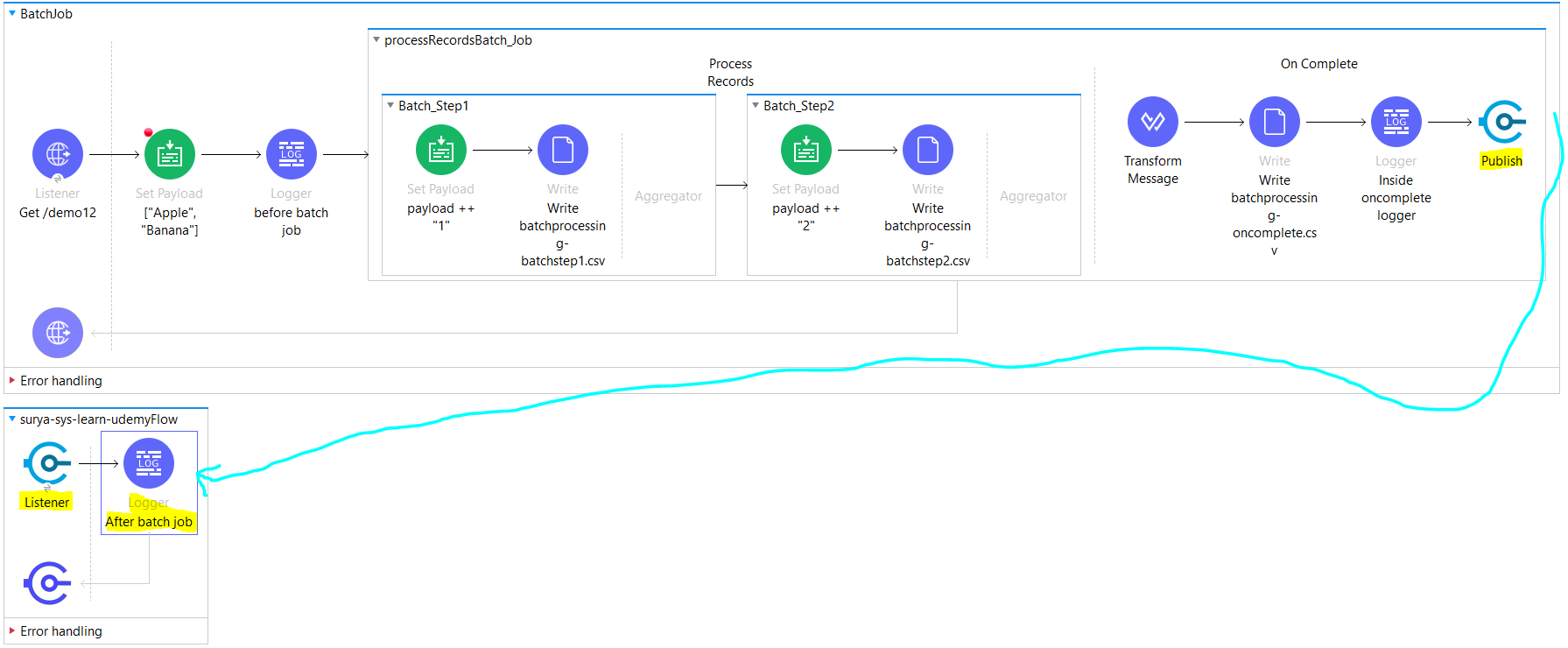
As we are aware that batch processing in mule is async i.e fire n forget so let say we want to do some logic after the batch job completes its processing so we can’t just only keep the any processeor after batch job definition like below img



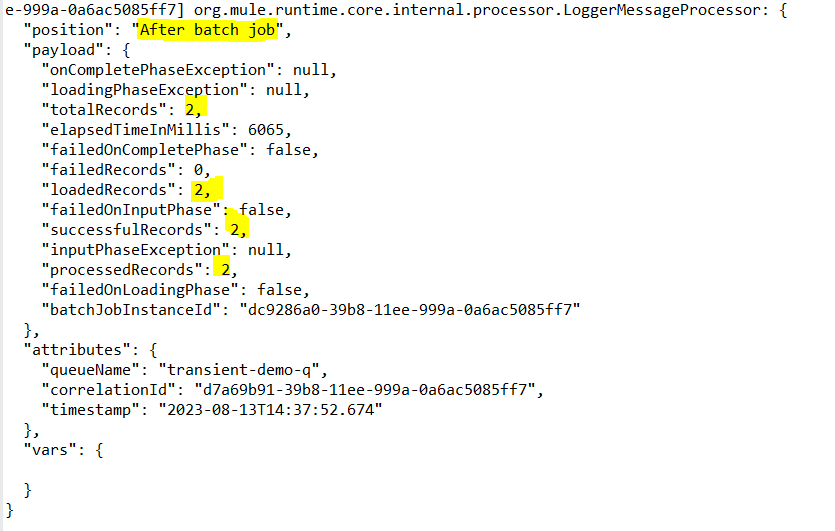
In above case How it will process is listener will start execution then set payload then before logger n then trigger will go into the batchjob (fires n forgets) n without waiting for the batch job completion it will go n execute after batch job logger where it will print wrong values so it is not correct . like below img

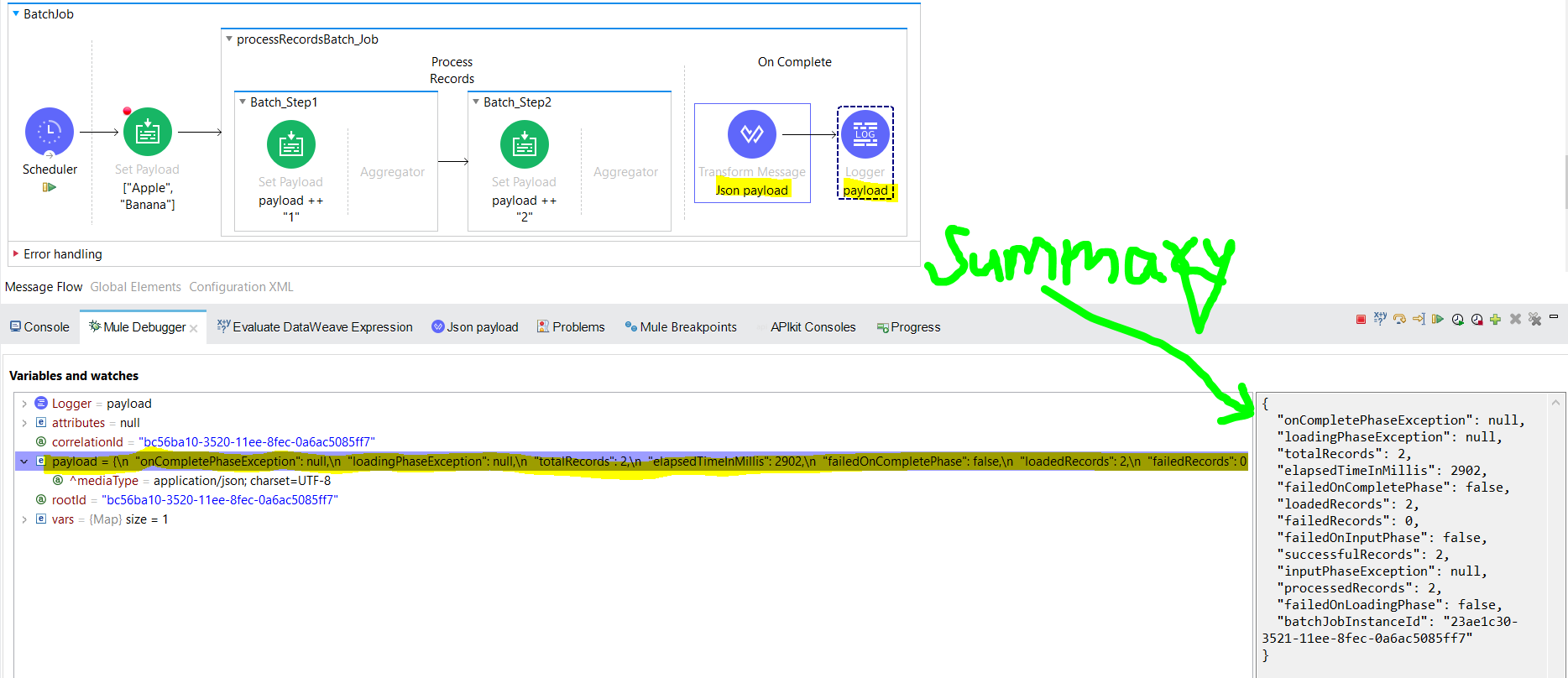


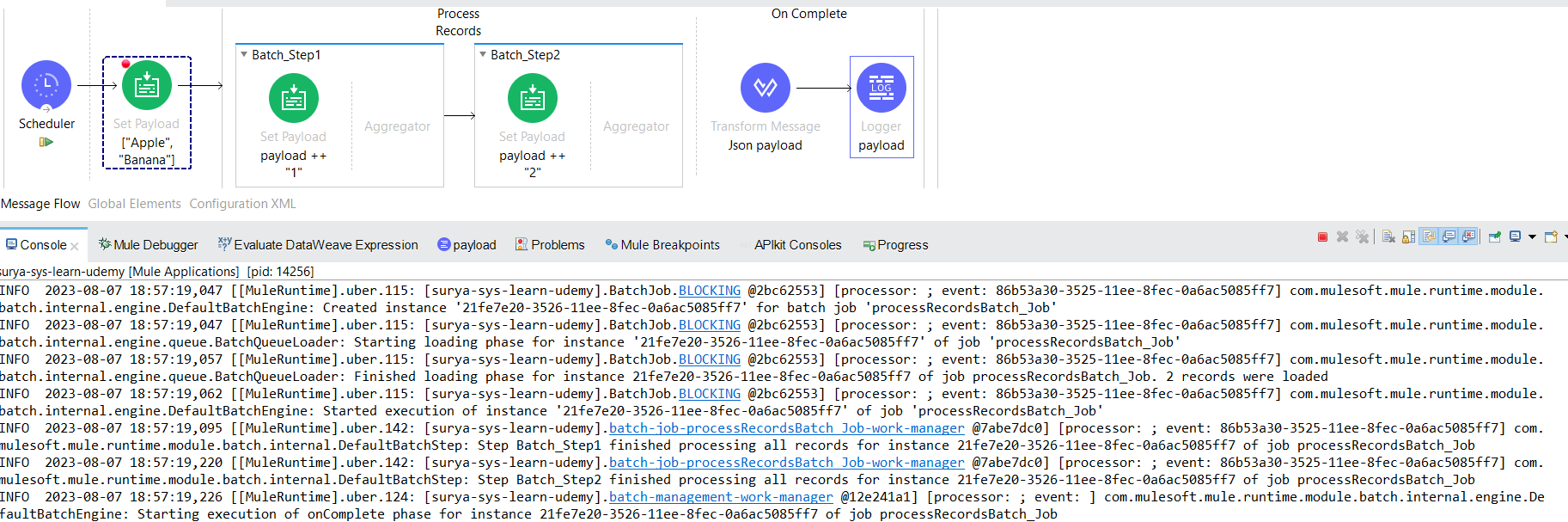
No, it is not possible to run a Batch synchronously within the flow it is called, by design. As an alternative you could put the logic that you want to execute after the batch in a separate flow that listens to a VM queue(VM Listener). In the On Complete phase of the batch you can send(VM Publish) a message to that VM queue. Like below image

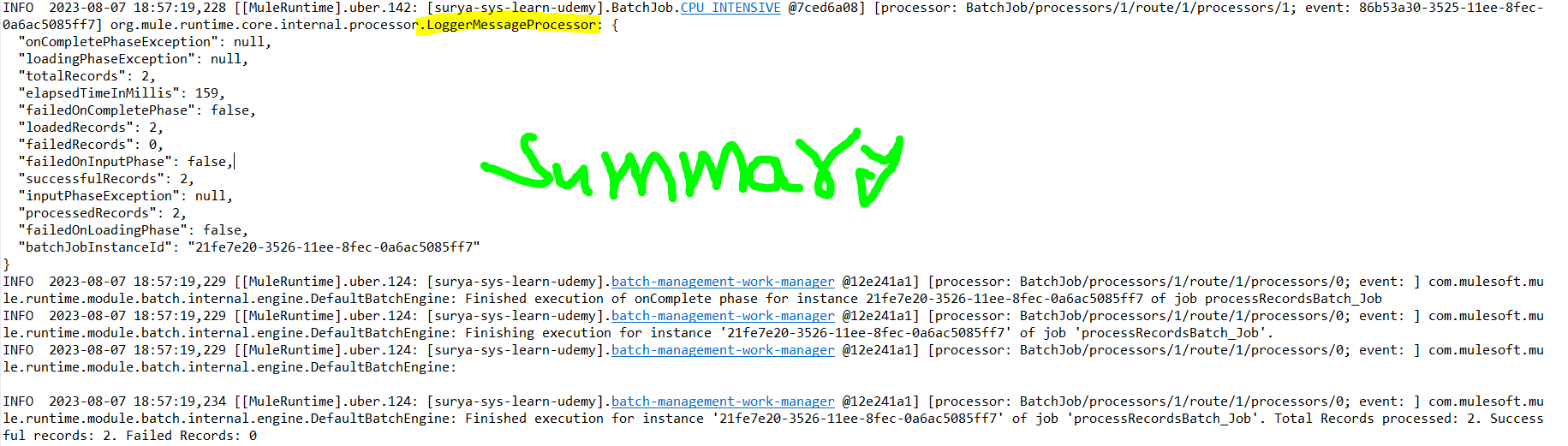


So you will achieve what u wanted to achieve like below img where we are logging after batch job looger after the atch job completes its execution.so will get correct response this time.









Scatter-Gather(Parallel Processing)

- Sends a request message to multiple targets concurrently. It collects the responses from all routes, and aggregates them into a single message.



|  |  |
| --- | --- |
| **Attributes** | |
| timeout | Sets the timeout for responses from sent messages, in milliseconds. A value of 0 or lower than 0 means no timeout. |
| maxConcurrency | Determines the maximum amount of concurrent routes to process. By default all routes run in parallel.  By setting this value to 1, scatter-gather processes the routes sequentially. |
| target | The name of the target variable. |
| targetValue | Value of the data to store in the target variable. If not set, the default value is #[payload]. This field accepts any value that a variable accepts:   * Any supported data type. * DataWeave expressions. * The keywords payload, attributes, and message, but *not* the keyword vars |

-The Scatter-Gather component is a routing event processor that processes a Mule event through different parallel processing routes that contain different event processors. Each route receives a reference to the Mule event and executes a sequence of one or more event processors. Each of these routes uses a separate thread to execute the event processors, and the resulting Mule event can be either the same Mule event without modifications or a new Mule event with its own payload, attributes, and variables. The Scatter-Gather component then combines the Mule events returned by each processing route into a new Mule event that is passed to the next event processor only after every route completes successfully.

-Scatter gather takes one mule event as input n that input gets processed in multiple routes and each route result is captured as payload in object format like below:

{

    "0": {

        "inboundAttachmentNames": [],

        "exceptionPayload": null,

        "inboundPropertyNames": [],

        "outboundAttachmentNames": [],

        "payload": "suryakant",

        "outboundPropertyNames": [],

        "attributes": {

            "headers": {

                "user-agent": "PostmanRuntime/7.32.3",

                "accept": "\*/\*",

                "cache-control": "no-cache",

                "postman-token": "2b5ea94b-25a8-4c0a-978b-da26b4fa6cd3",

                "host": "localhost:8081",

                "accept-encoding": "gzip, deflate, br",

                "connection": "keep-alive"

            },

            "clientCertificate": null,

            "method": "GET",

            "scheme": "http",

            "queryParams": {

                "name": "suryakant",

                "age": "29",

                "company": "Deloitte"

            },

            "requestUri": "/surya-sys-learn-udemy/demo7?name=suryakant&age=29&company=Deloitte",

            "queryString": "name=suryakant&age=29&company=Deloitte",

            "version": "HTTP/1.1",

            "maskedRequestPath": null,

            "listenerPath": "/surya-sys-learn-udemy/demo7",

            "relativePath": "/demo7",

            "localAddress": "/127.0.0.1:8081",

            "uriParams": {},

            "rawRequestUri": "/surya-sys-learn-udemy/demo7?name=suryakant&age=29&company=Deloitte",

            "rawRequestPath": "/surya-sys-learn-udemy/demo7",

            "remoteAddress": "/127.0.0.1:53281",

            "requestPath": "/surya-sys-learn-udemy/demo7"

        }

    },

    "1": {

        "inboundAttachmentNames": [],

        "exceptionPayload": null,

        "inboundPropertyNames": [],

        "outboundAttachmentNames": [],

        "payload": "kashinath",

        "outboundPropertyNames": [],

        "attributes": {

            "headers": {

                "user-agent": "PostmanRuntime/7.32.3",

                "accept": "\*/\*",

                "cache-control": "no-cache",

                "postman-token": "2b5ea94b-25a8-4c0a-978b-da26b4fa6cd3",

                "host": "localhost:8081",

                "accept-encoding": "gzip, deflate, br",

                "connection": "keep-alive"

            },

            "clientCertificate": null,

            "method": "GET",

            "scheme": "http",

            "queryParams": {

                "name": "suryakant",

                "age": "29",

                "company": "Deloitte"

            },

            "requestUri": "/surya-sys-learn-udemy/demo7?name=suryakant&age=29&company=Deloitte",

            "queryString": "name=suryakant&age=29&company=Deloitte",

            "version": "HTTP/1.1",

            "maskedRequestPath": null,

            "listenerPath": "/surya-sys-learn-udemy/demo7",

            "relativePath": "/demo7",

            "localAddress": "/127.0.0.1:8081",

            "uriParams": {},

            "rawRequestUri": "/surya-sys-learn-udemy/demo7?name=suryakant&age=29&company=Deloitte",

            "rawRequestPath": "/surya-sys-learn-udemy/demo7",

            "remoteAddress": "/127.0.0.1:53281",

            "requestPath": "/surya-sys-learn-udemy/demo7"

        }

    },

    "2": {

        "inboundAttachmentNames": [],

        "exceptionPayload": null,

        "inboundPropertyNames": [],

        "outboundAttachmentNames": [],

        "payload": "Dubalgunde",

        "outboundPropertyNames": [],

        "attributes": {

            "headers": {

                "user-agent": "PostmanRuntime/7.32.3",

                "accept": "\*/\*",

                "cache-control": "no-cache",

                "postman-token": "2b5ea94b-25a8-4c0a-978b-da26b4fa6cd3",

                "host": "localhost:8081",

                "accept-encoding": "gzip, deflate, br",

                "connection": "keep-alive"

            },

            "clientCertificate": null,

            "method": "GET",

            "scheme": "http",

            "queryParams": {

                "name": "suryakant",

                "age": "29",

                "company": "Deloitte"

            },

            "requestUri": "/surya-sys-learn-udemy/demo7?name=suryakant&age=29&company=Deloitte",

            "queryString": "name=suryakant&age=29&company=Deloitte",

            "version": "HTTP/1.1",

            "maskedRequestPath": null,

            "listenerPath": "/surya-sys-learn-udemy/demo7",

            "relativePath": "/demo7",

            "localAddress": "/127.0.0.1:8081",

            "uriParams": {},

            "rawRequestUri": "/surya-sys-learn-udemy/demo7?name=suryakant&age=29&company=Deloitte",

            "rawRequestPath": "/surya-sys-learn-udemy/demo7",

            "remoteAddress": "/127.0.0.1:53281",

            "requestPath": "/surya-sys-learn-udemy/demo7"

        }

    }

}

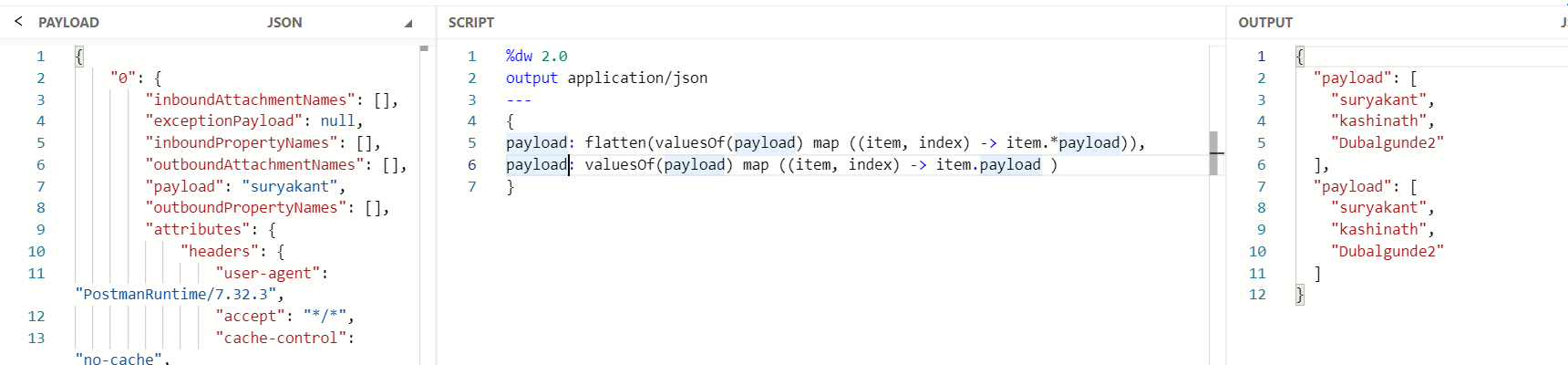
- To extract all resulting payloads to an array, use a DataWeave script, such as the following one:

flatten(valuesOf(payload) map ((item, index) -> item.\*payload))

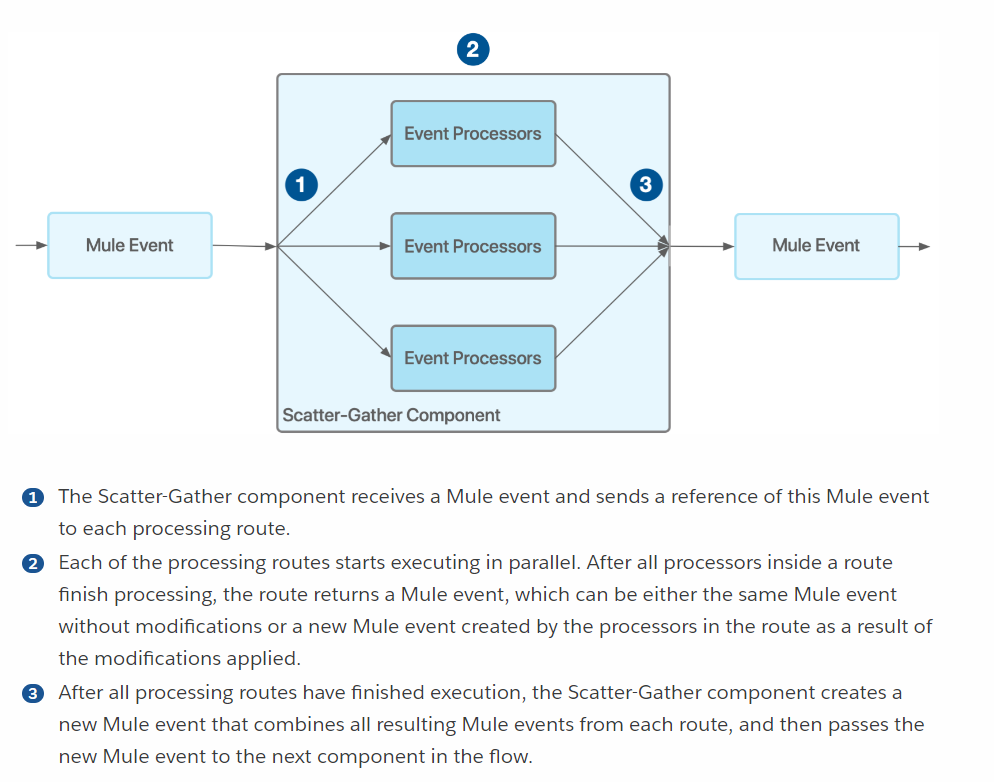
or

valuesOf(payload) map ((item, index) -> item.payload )

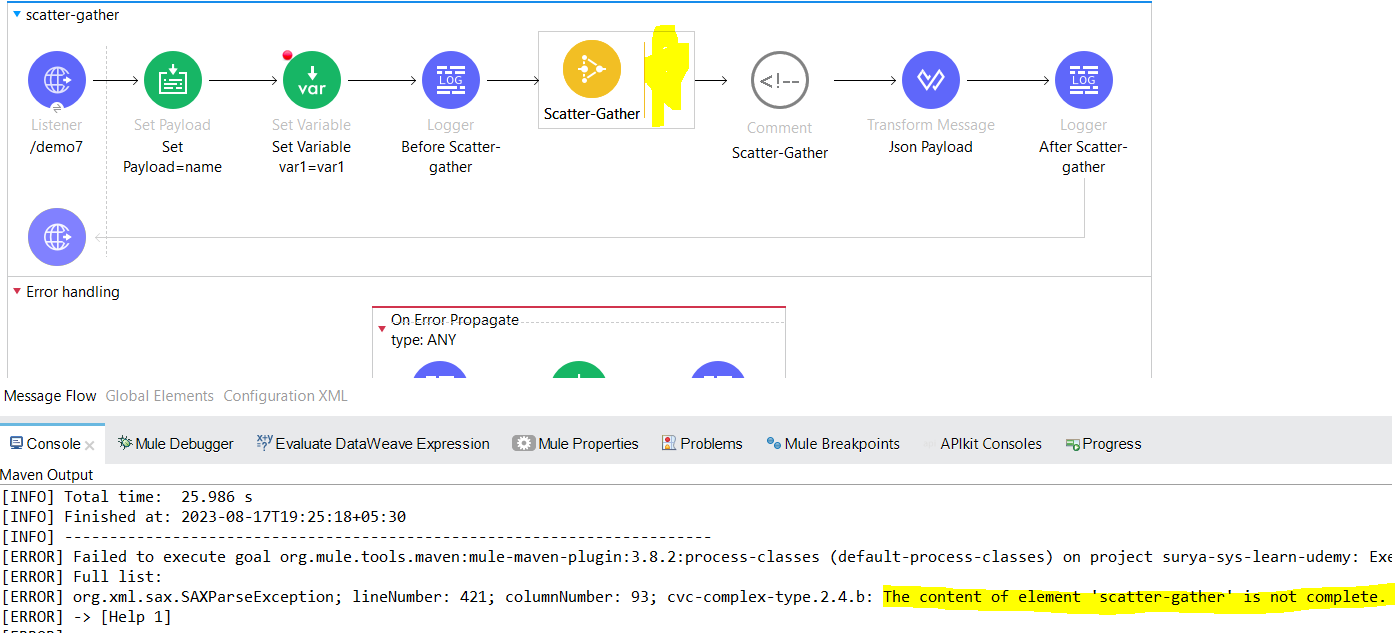
both provides same output



- The Scatter-Gather component executes each route in parallel, not sequentially.



-Note: Configure your Scatter-Gather component to have at least two routes; otherwise, your Mule application throws an exception like below and will fail at build step.



we should have more than one route defined in scatter gather

## **Variable Propagation**

Every route starts with the same initial variable values. Modifications to a variable within a specific route do not affect other routes. So, if a variable is added or modified in one route, then, after aggregation, the value is defined by that route. If a variable is added or modified by more than one route, the value is added to a list of all the values defined for that variable within all the routes, for example:

<set-variable variableName="var1" value="var1"/>

<set-variable variableName="var2" value="var2"/>

<scatter-gather doc:name="Scatter-Gather" doc:id="abc665e0-6119-4ecb-9f8b-52dbcbb1d488" >

<route >

<set-variable variableName="var2" value="newValue"/>

<set-variable variableName="var3" value="appleVal"/>

</route>

<route >

<set-variable variableName="var3" value="bananaVal"/>

</route>

<route >

<set-variable variableName="var3" value="otherVal"/>

<set-variable variableName="var4" value="val4"/>

</route>

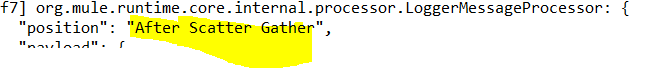
</scatter-gather>

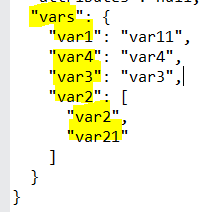
After aggregation, the variables are:

{var1: "var1", var2: "newValue", var3: ["appleVal, bananaVal, otherVal"], var4: "val4"}

-variables declared before scatter gather can be accessible inside scatter gather + new variables can be created n accessed along with variables declared before scatter gather once scatter gather completes.







## Error Handling Inside Scatter-Gather Routes

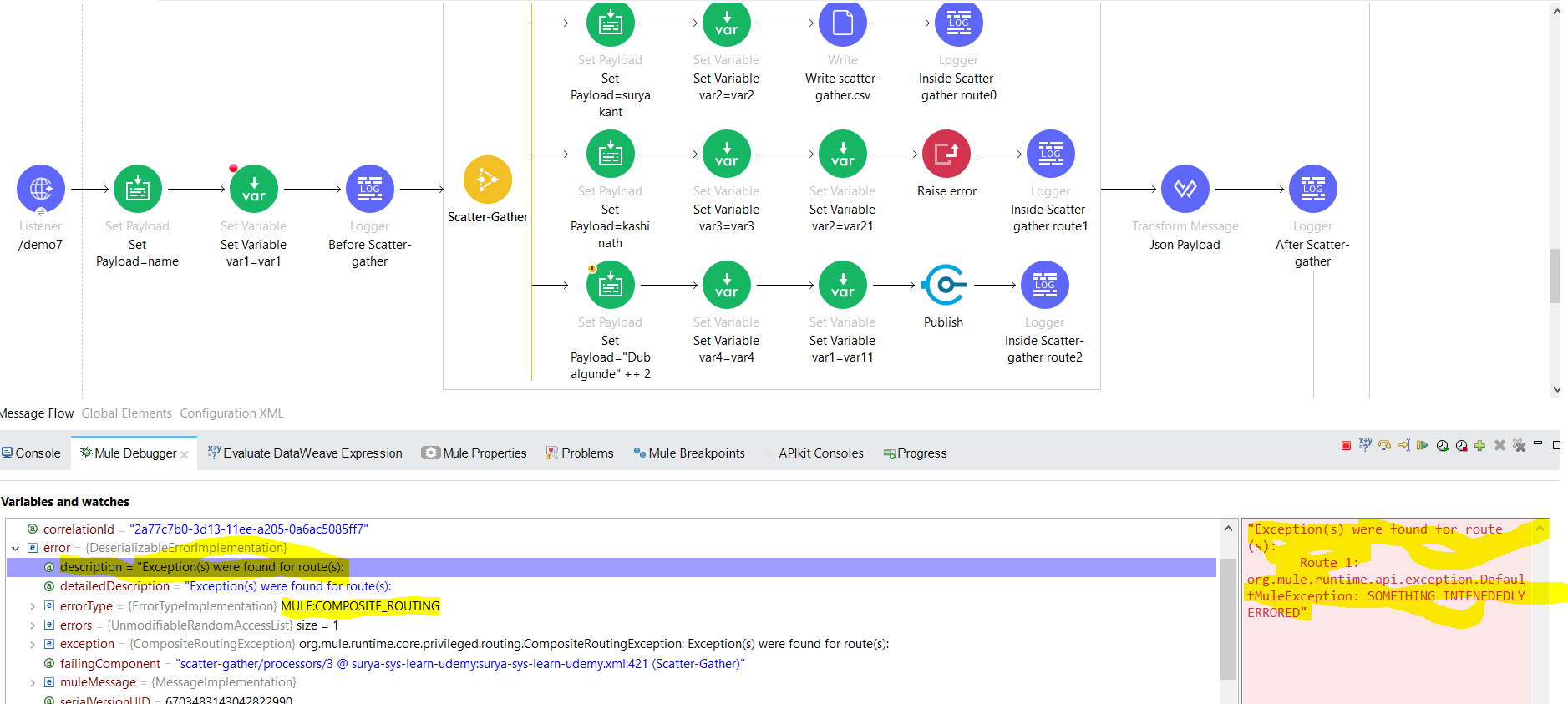
You can use a Try scope in each route of a Scatter-Gather component to handle any errors that might be generated by a route’s event processors. After every route executes, if any has failed with an error, then the Scatter-Gather component throws an error of type MULE:COMPOSITE\_ROUTING, and event processing does not proceed past the Scatter-Gather component in the flow. Instead, the flow branches to your error-handling event processors.

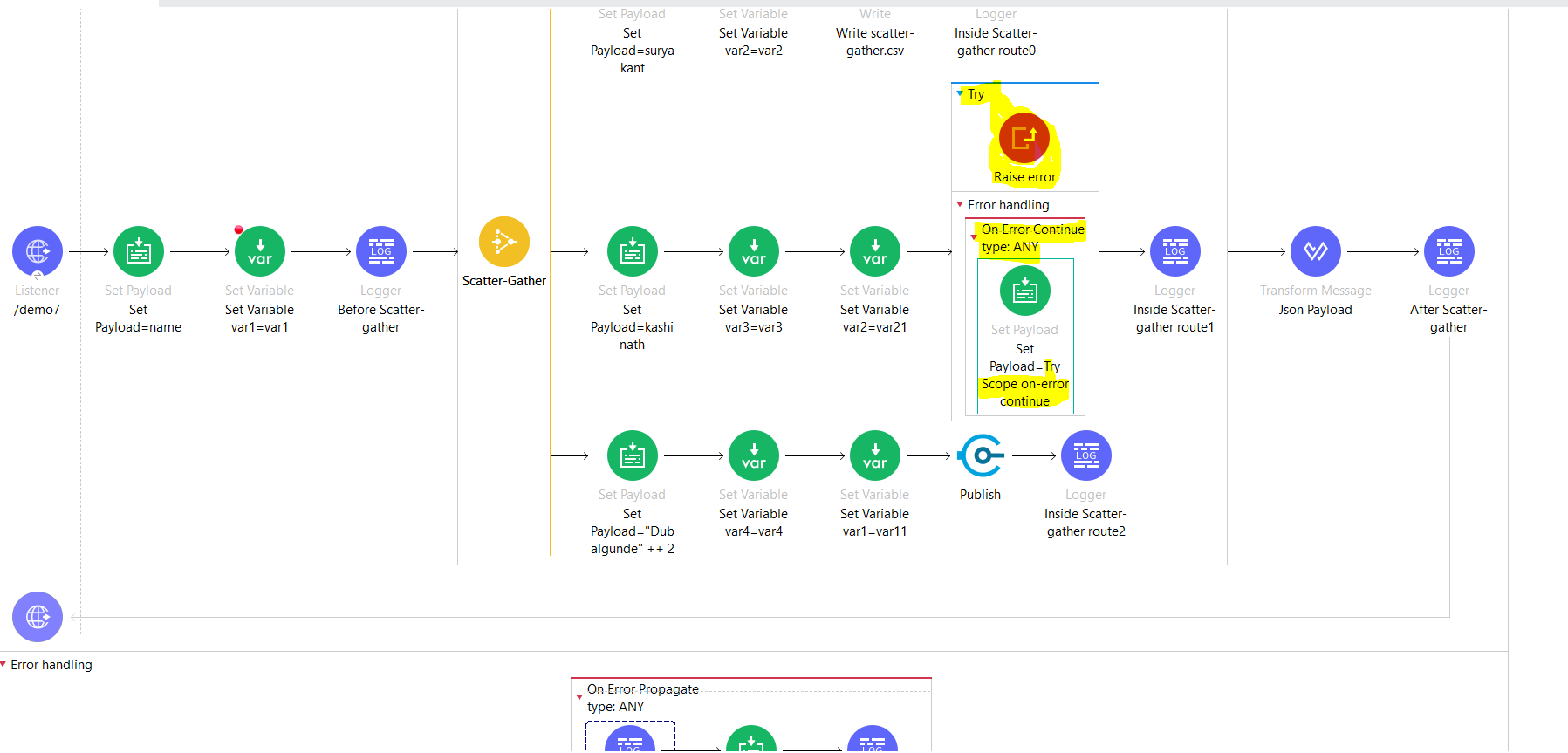
Because the MULE:COMPOSITE\_ROUTING error object gathers not only errors from routes that failed but also Mule events from successfully completed routes, your application can use the error-handling event processors to process Mule events from the routes that completed

To illustrate how this works, consider the following two cases:

* The routes in a Scatter-Gather component each contain a Try scope.  
  One of the routes generates an error that is successfully handled by that route’s Try scope through an on-error-continue error handler, so the route is completed successfully. The Scatter-Gather component consolidates the Mule events from all routes into a new Mule event and passes the consolidated event to the next event processor.
* One of the routes in a Scatter-Gather component does not contain a Try scope or contains a Try scope with an error handler that cannot handle the error type, or the error handler is an on-error-propagate type.  
  An error occurs in this route, causing the route to fail, which in turn causes the Scatter-Gather component to throw a MULE:COMPOSITE\_ROUTING error. The flow branches to your error-handling event processors, which are able to process the Mule events from the completed routes.

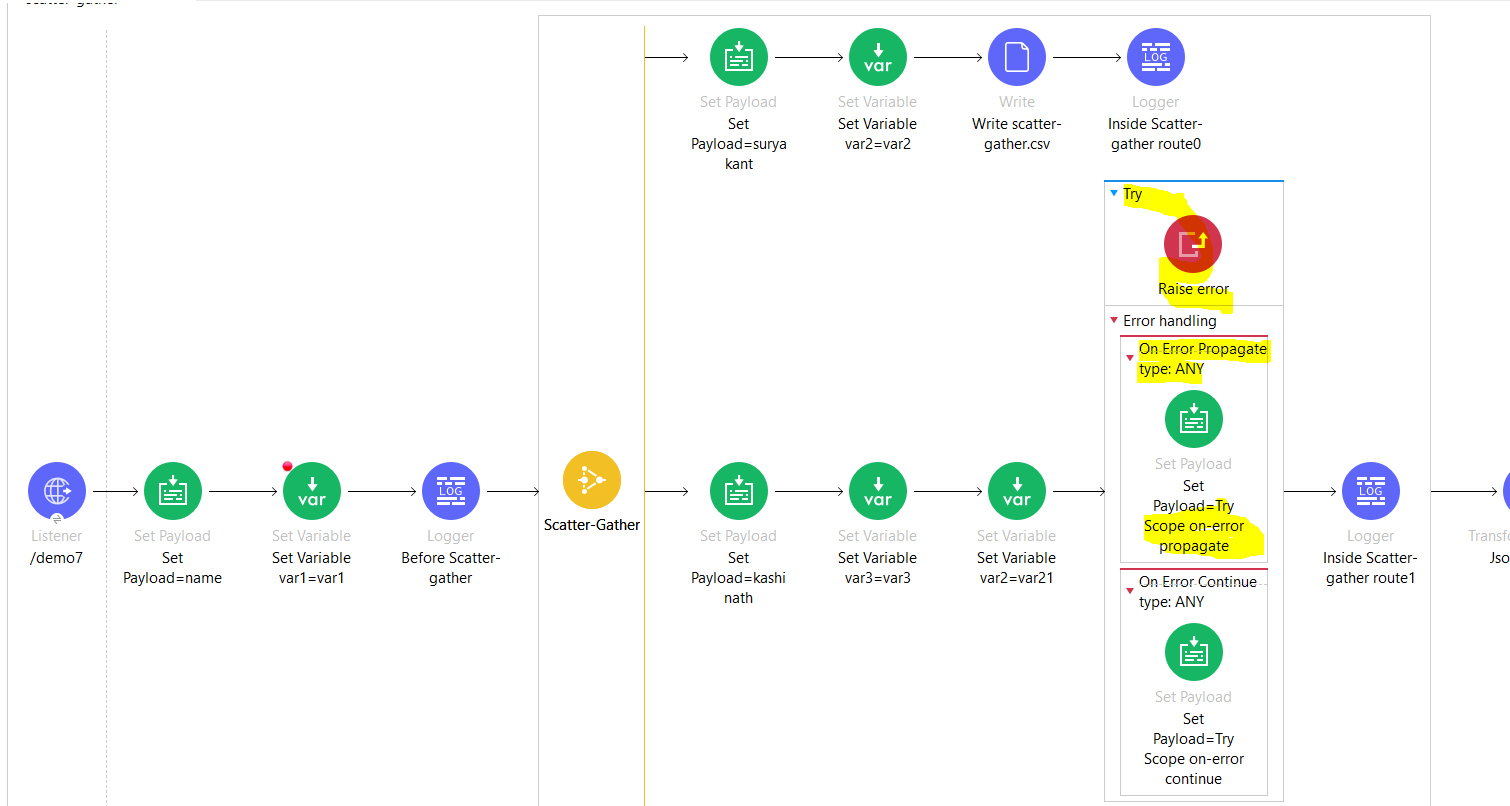






-if we use try scope on raise error then when trigger goes to the raise error it errors out n since we have specified the error handler as on-error-continue so it will treat as success n will continue to process further but incae if we configured on-error propagate in try scope then it will treat as failure and will give MULE:COMPOSITE\_ROUTING erro





Example of handling these errors:

<flow name="errorHandler">

<scatter-gather>

<route>

<raise-error type="APP:MYERROR"/>

</route>

<route>

<set-payload value="apple"/>

</route>

</scatter-gather>

<error-handler>

<on-error-continue type="MULE:COMPOSITE\_ROUTING">

<!-- This will have the error thrown by the first route -->

<logger level="WARN" message="#[error.errorMessage.payload.failures['0']]"/>

<!-- This will be a null value -->

<logger level="WARN" message="#[error.errorMessage.payload.failures['1']]"/>

<!-- This will be a null value -->

<logger level="WARN" message="#[error.errorMessage.payload.results['0']]"/>

<!-- This will have the result of the second (correctly executed) route -->

<logger level="WARN" message="#[error.errorMessage.payload.results['1']]"/>

</on-error-continue>

</error-handler>

</flow>

## **Handle Timeout Errors in a Scatter-Gather**

If you configure a timeout for a Scatter-Gather component and a route does not complete processing before the timeout expires, the route throws a MULE:TIMEOUT error. This error is then handled the same way as any other error generated from a route: after each route completes (either by processing success or by throwing a MULE:TIMEOUT error), the successful results and errors are collected together in the Scatter-Gather component MULE:COMPOSITE\_ROUTING error, which is then processed in your configured error handler.

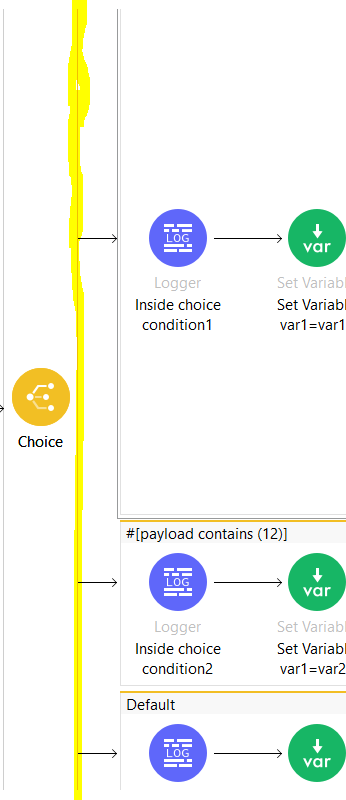
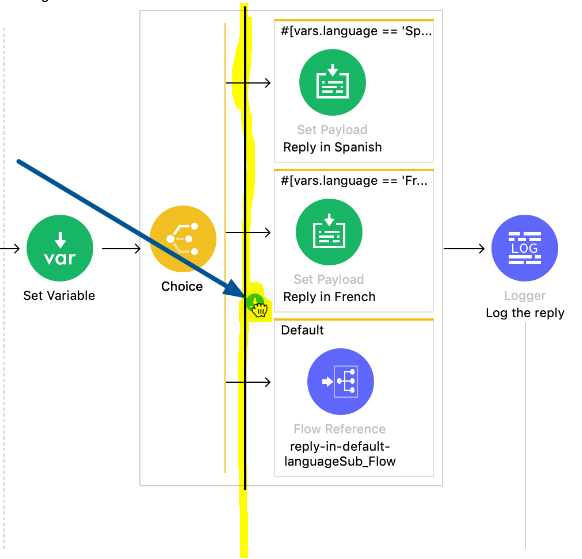
Choice Router: (If-else)

-The Choice router adds conditional processing to a flow, similar to an if/then/else code block in most programming languages.

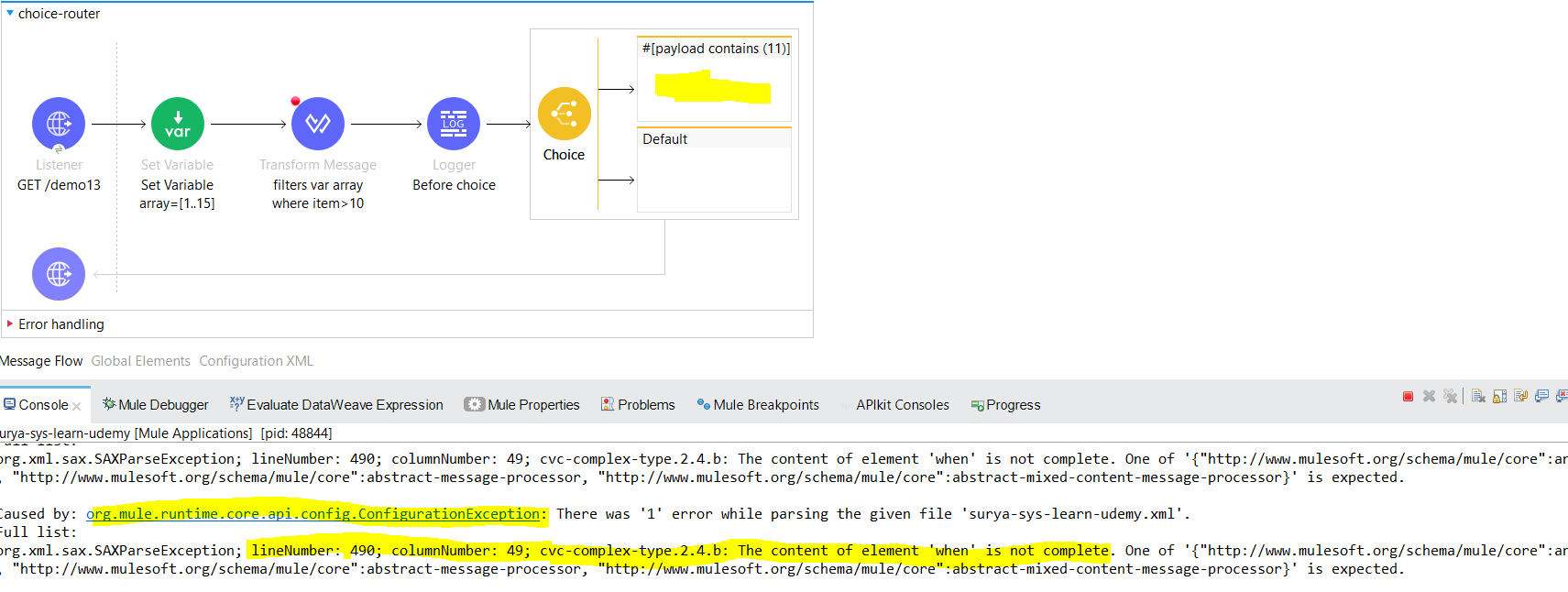
-The Choice router dynamically routes messages to different portions of a flow according to a set of DataWeave expressions that evaluate message content. Each expression is associated with a different routing option, and the Default case executes when no other expressions match.

-Only one of the routes in the Choice router executes, meaning that the first expression that evaluates to true triggers that route’s execution and the others are not checked. If none of the expressions are true, then the default route executes.

-By default, when we drag choice router in the canvas it comes with one when condition n default condition, to add more when condition drag any processor like logger or transform message towards the yellow straight line of choice router once u can see the yellow line became black just drop ur processor there so it will add a new when condition

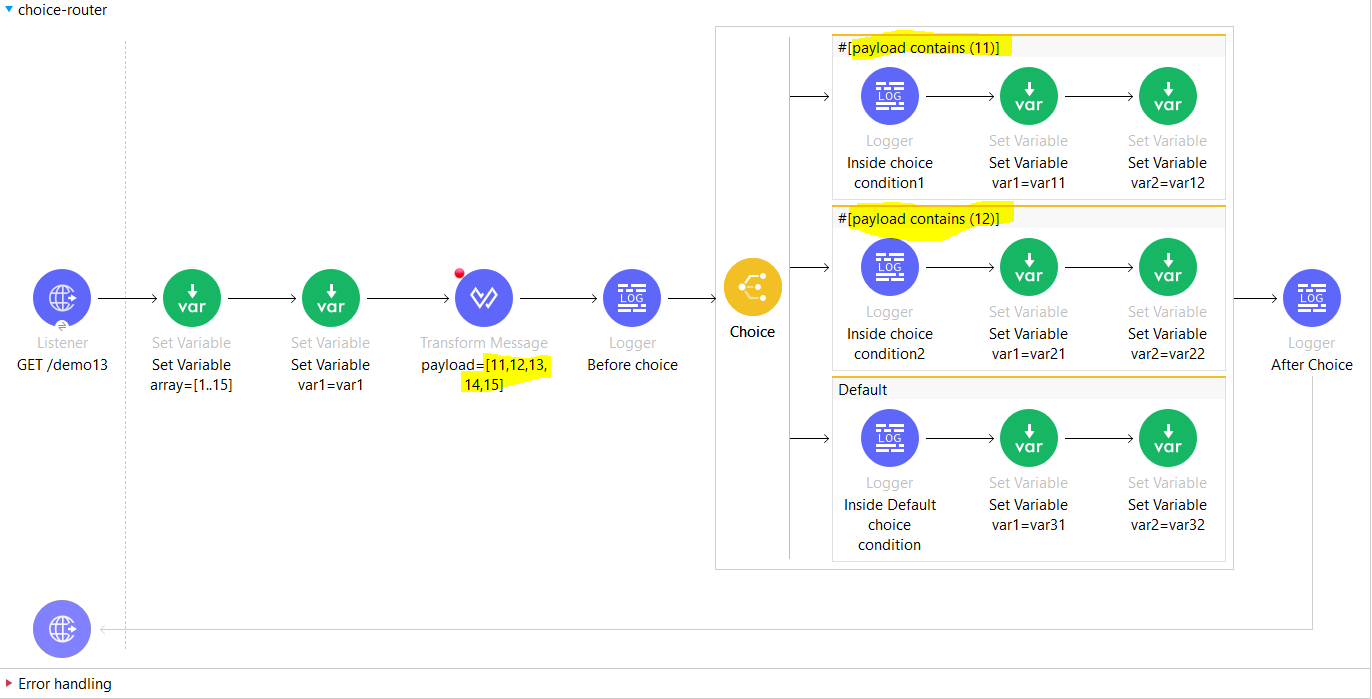
 

-the when condition should have atleast 1 event processor declared in it.otherwise will get below error



-Default case can be empty.it will work

-In case we have a scenario wherewe have 2 choice when conditions n both are true n matching so choice will always give priority to the 1st when condition as it starts checking the when condition from upper to till default.. like below scenario



-variables declared before choice can be accessible inside choice + new variables can be created n accessed before choice once choice completes.

First-Successful:

- 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), and trigger moves to the next configured route.

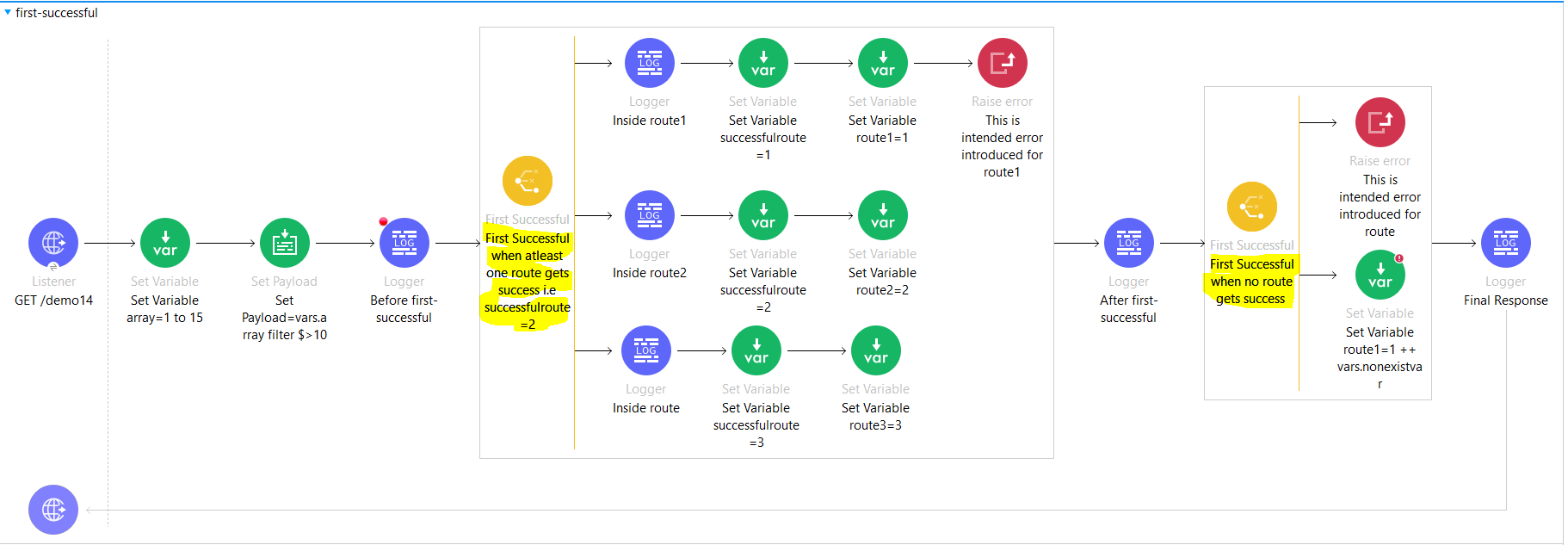
If none of the configured routes execute successfully, the First Successful router throws an error.

-By default when we drag-n-drop the first successful router into canvas it comes with no routes i.e (project fails to deploy as it throws error The content of element 'first-successful' is not complete.) to add route we can drag any processor like logger or transform message towards the yellow straight line of router once u can see the yellow line became black just drop ur processor there so it will add a new route…atleast we should have 1 route defined for first successful.

-In the below example it explains both the scenarios

1) when we have successful route in first successful router---1st first successful router

2) when we dont have successful route in first successful router---2nd first successful router



1st first successful router: here it will start with route 1 prints logger ,sets var successfulroute to 1 n route1=1 but next processor hs error raised so that means this route is not successful so it will throw the error n will trigger route 2 where it prints logger, sets var successfulroute to 2 n route2=2 since all processors in the route 2 has processed successfully so it will come out of the 1st first successful router n execute after first successful logger by ignoring route 3 as it name suggest it will try to find first successful route n since it got its first successful route as route 2 so it will ignore route 3..

Note: since it got its first successful route as route 2 while accessing vars in after successful logger we will get only 3 vars i.e array,successfulroute=2 n route2=2 it will not be able to access the variables declared inside route 1 i.e successfulroute=1 n route1=1

2nd first successful router: here it will start with route 1 where we have raise error component declared so it will trigger route 2 n here we are trying to concatenate 1 ++ with nonexisting var so this route also is not successful so in this case where none of the route is successful ,2nd first successful router will throw the error the error will be the last route’s error since in the above diagram we have set var route1=1++ vars.nonexistvar where it throws MULE:EXPRESSION error

""You called the function '++' with these arguments: numer + null “ n stops the execution of the flow n handles the error since no component,no flow ,no global error handler so it will handle thru default error handler which provides on error propagate 500 status code with error.description

Round-Robin:

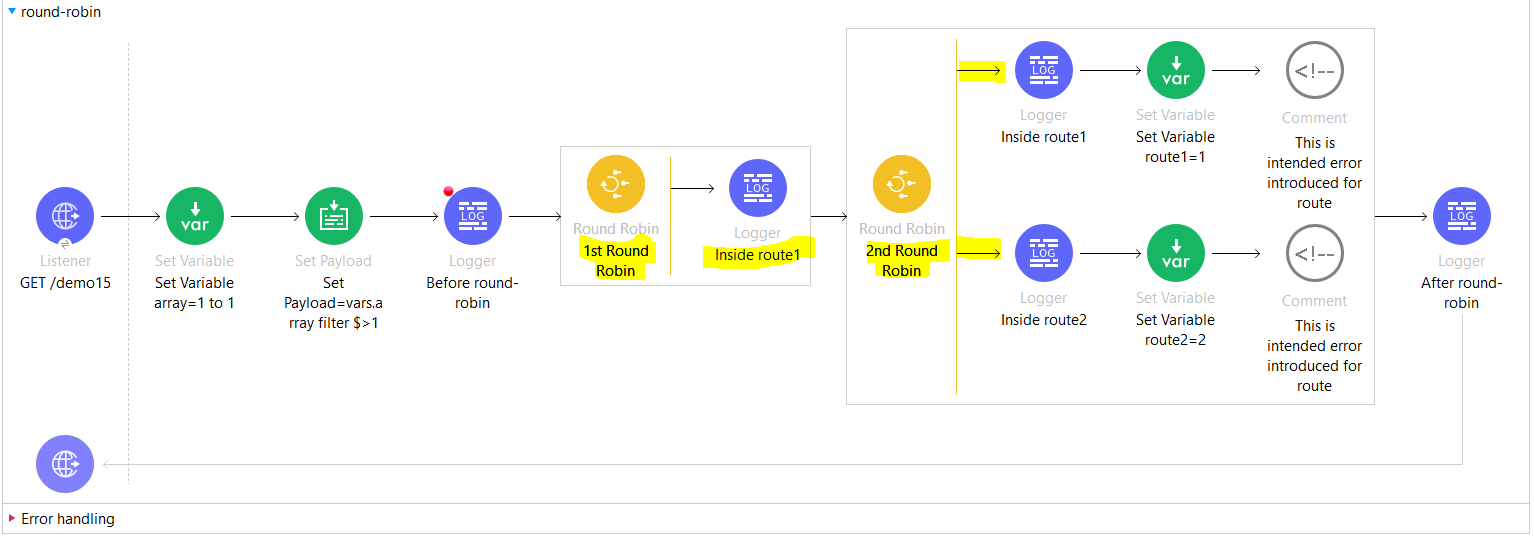
- Round Robin is a Mule component that contains one or more processing routes and executes only *one* of the routes each time the flow executes the component.

-Starts with the first route, Round Robin initiates execution of processors within the route over a Mule event and sends the results to the next processor in the flow. Round Robin keeps track of the previously selected route and never selects the same route consecutively.

For example, the first time Round Robin executes, first route components. The next time, it selects the second route. If the previously selected route is the last route, Round Robin executes the first route.n if we have only 1 route configured so it will execute route 1 eveytime.

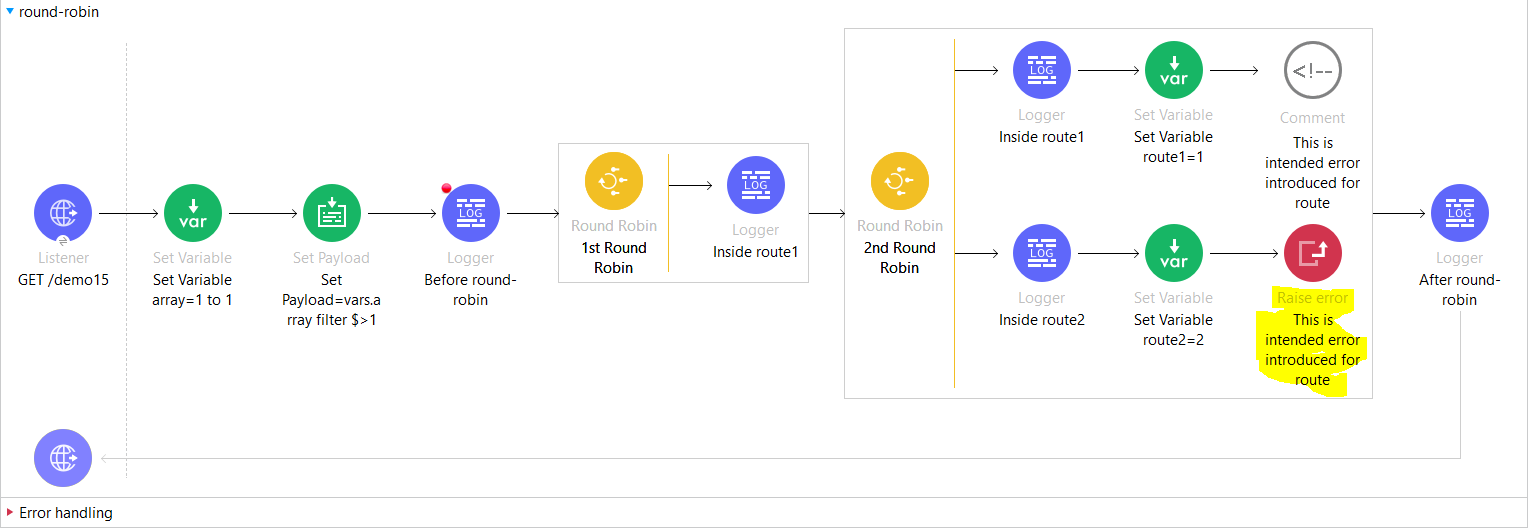
-The payload, variables, and attributes from the Mule event that reaches Round Robin are accessible for processing within the route. Any changes to the Mule event within a route propagate to the processor after Round Robin.

- By default when we drag-n-drop the round-robin router into canvas it comes with no routes i.e (project fails to deploy as it throws error The content of element 'round-robin' is not complete to add route we can drag any processor like logger or transform message towards the yellow straight line of router once u can see the yellow line became black just drop ur processor there so it will add a new route…atleast we should have 1 route defined for round-robin.



* In the above diagram we have 2 round robin

1st round-robin has only 1 route so every time it selects route 1 onlyn loggs inside route1 logger then it will trigger 2nd round robin where we have 2 routes so first time it will execute route 1 n once route 1 processed successfully then prints after round-robin logger when again the flow get triggered it will go into 1st round-robin n execute inside route1 logger then then it will trigger 2nd round robin where it selects 2nd route to execute n after successful processing it prints after round-robin logger.



-in the above ss,we have introduced raise error in 2nd round robin’s 2nd route so in this case also round robin will initially executes 1st route n in next iteration it goes to 2nd route n then again route 1 ,route 2,route 1….. goes on.so what does that means is round robin will never ignore the route even it has error it will execute sequentially like route 1 ,route 2,route1,route2 ……

-Throws MULE:ROUTING error.

Cache scope:

-used for storing and reusing frequently called data.

-The content of element 'ee:cache ' is not complete. Will get if no processor is defined inside cache scope.

-You can use a Cache scope to reduce the processing load on the Mule instance and to increase the speed of message processing within a flow.

- When a message processor in a Mule app sends a message to the Cache scope, the parent flow expects an output. The Cache scope processes the message payload, delivers the output to the parent flow, and saves the output (that is, caches the response). The next time the Cache scope receives the same kind of message payload, the scope can offer a cached response rather than invoking a potentially time-consuming process again.

For instance, you can use a Cache scope to manage customer requests for flight information. Many customers might request the same pricing information about flights from San Francisco to Buenos Aires. Rather than using a lot of processing power to send separate requests to several airline databases with each customer query, you can use a Cache scope to arrange to send a request to the databases fewer times – say, once every ten minutes – and present users with the cached flight pricing information. Where timeliness of data is not critical, Cache scope can save time and processing power.

Note that you can put any number of message processors (such as connectors or components) into a Cache scope and configure the caching strategy to store the responses (which contain the payload of the response message) produced by the processing that occurs within the scope.

- By default, the Cache scope uses a caching strategy that stores data in an in-memory object store. You can create a custom caching strategy that references an existing object store, or you can create a new custom object store to use in this caching strategy.

- Instead of processing all message payloads that it receives, the Cache scope can exclude specific payloads from the Cache scope flow based on an DataWeave expression. This we can specify in Filter config of cache scope.

Opt to set up a filter for specific payloads if you need one.

Example that uses a DataWeave expression: filterExpression="#[user.isPremium()]"

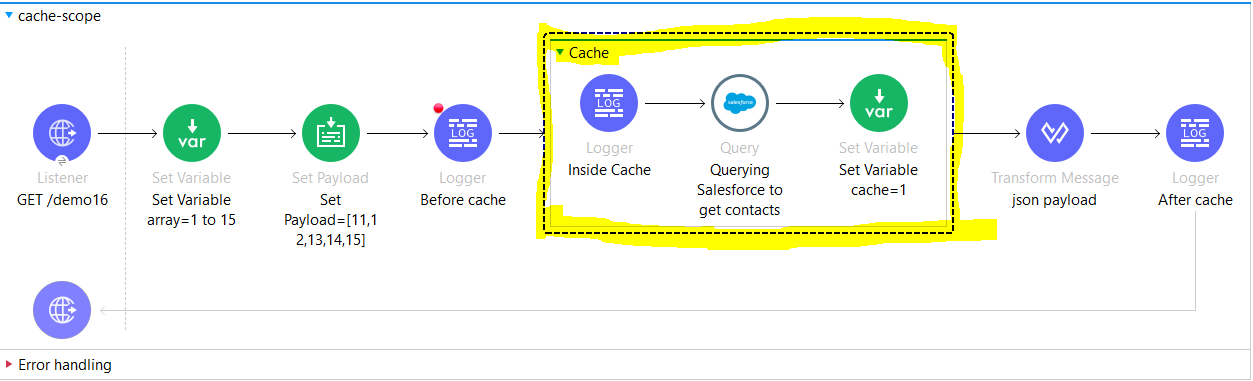
If the message matches the expression(s), Mule executes the Caching Strategy.

If the message does not match expression(s), Mule processes the message through all message processors within the Cache scope but never saves or produces cached responses.

How cache scope works is:-As definition it is used to store n reuse the frequently accessed data so in below ss: cache scope stores the payload that is generated by processor inside it.

when first time we call /demo16 endpoint thru postman or any rest client or browser it will process in order like set variable array,set paylaod to [11,12,13,14,15],logs before cache logger,will go inside cache scope n executes complete scope prints inside cache logger,queries salesforce,set var cache=1 n then it converts the response to json n prints after cache logger

The next time the Cache scope receives the same kind of message payload, the scope can offer a cached response rather than invoking a potentially time-consuming process again i.e. it wont go inside the cache scope n processes all the processors defined inside cache scope .as shown in the ss when trigger will come to cache scope it will ignore the processors inside the cache scope n provides the response that is stored as cached response n will directly executes the next processor that converts the java into json.



-The cached scope is not useable in case we want to filter the records like with queryparams for eg: let say at first time we have given queryparam name=siddhartha n we are using this queryparam in sfdc query to filter the records where name=siddhartha so since it is intial call so trigger will go inside cache scope n will execute all the processors inside it like logs inside cache logger,query n filters the records within sfdc by queryparam name=siddhartha n set var cache=1 n then it will convert the siddhartha record payload in json n prints after cache logger

But next time if we change the queryparam name=avi or anything apart from siddhartha then also we will get the payload from cached response as when trigger comes to cache scope it wont go inside the cache scope n execute the processors again but directly gives the cachedresponse from cahe response so in this case it is not usable.

TRY Scope:

- The Try scope enables you to handle errors that may occur when attempting to execute any of the components inside the Try scope. It also supports transactions. A Try scope wraps one or more operations, then catches and handles any exceptions that might be thrown by any of these enclosed operations.

- The Try scope catches and handles errors that occur when executing any of the components placed inside.

-The components inside the Try scope can also be configured as a Transaction via the "Transactional Action" property, in which case they will never be partially executed. Every operation within the scope of a transaction is executed in the same thread, and errors should lead to either a rollback or a commit.

- The content of element 'try' is not complete. When we don’t define any processor inside try scope

## **Error Handling with the Try Scope**

When designing your flow, try to group those operations that are likely to experience errors inside a Try scope. The Try scope enables you to isolate potentially troublesome operations in your flow and assign them an error-handling method. You can also configure the operations inside the Try scope to be processed as a transaction.

The Try scope has an error handling strategy that you configure in the same way you configure error handling for a flow.

The Try scope can distinguish among various error type conditions and apply different behaviors. If an error is raised by a component inside a Try scope, then the Try scope’s error handler is executed. At this point, the error is available for inspection, so the handlers can execute and act accordingly:

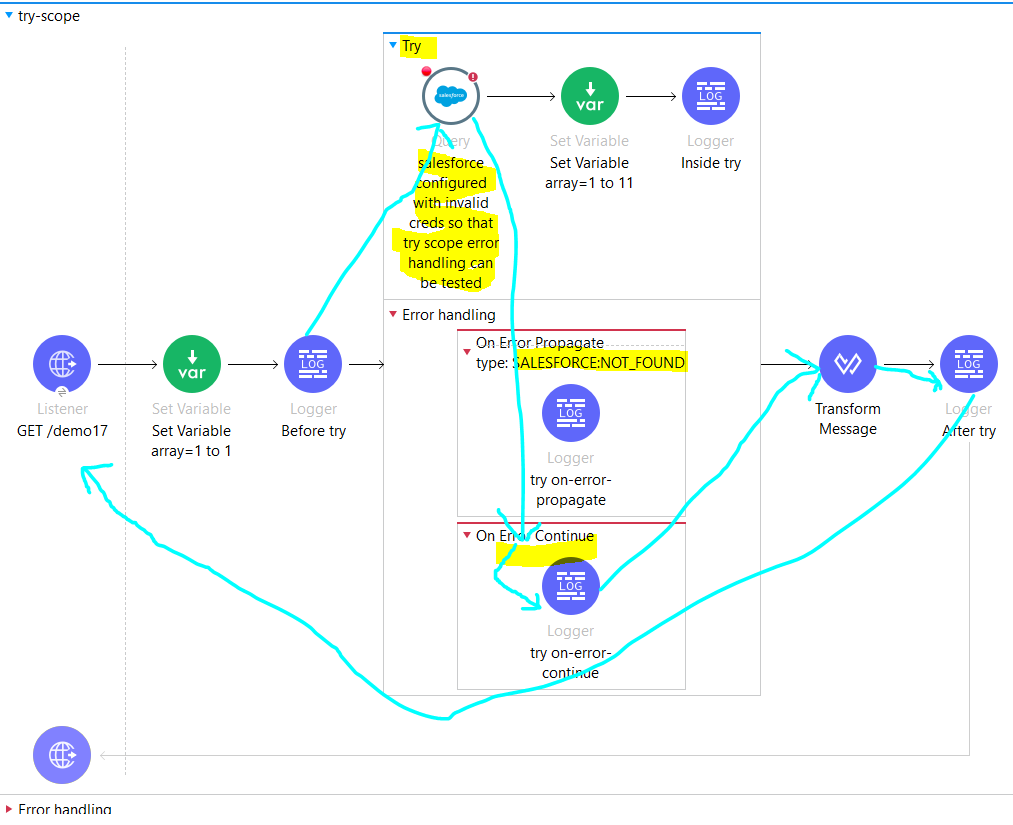
* On Error Continue

Executes and sends the result of the execution to its container Try scope, which uses that result to complete the execution successfully. Any transactions at this point are also committed.

* On Error Propagate

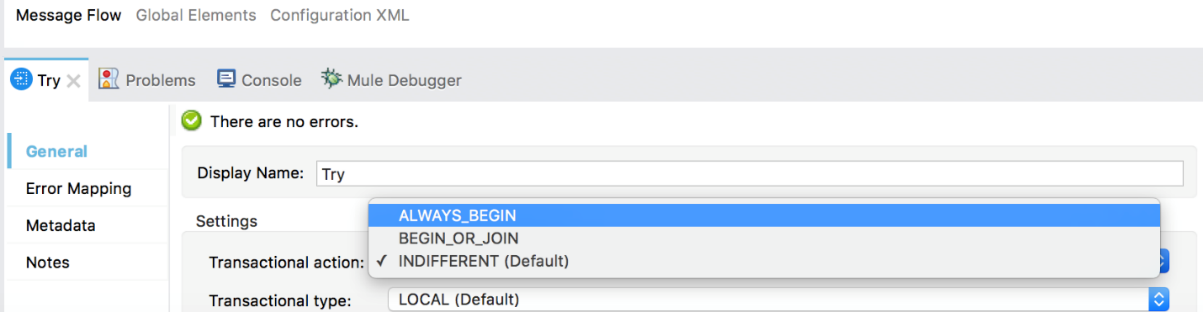
Rolls back any transactions, then executes and uses that result to re-throw the existing error, causing its container Try scope’s execution to fail.

- If the Try scope has several components, then after a component raises an exception, subsequent components in the Try scope are not executed, regardless of the type of error handler that catches the exception. In the case of On Error Propagate, the error is propagated to the flow’s error handler, as if the Try scope did not exist. In the case of On Error Continue, processing continues outside the Try scope at the next flow component, as if the Try scope never threw an exception.



## Handling Transactions

A transaction is a series of actions that should never be partially executed. Configure a Try scope so that it is a set of operations that are considered one unit that either succeeds or fails, depending on whether errors are propagated and the transaction rolled back, or handled and the transaction committed. In either case, the process flow in which the Try scope resides continues.



The Try scope treats child operations as a transaction when the Transactional Action (transactionalAction) is set to ALWAYS\_BEGIN or BEGIN\_OR\_JOIN. It can be configured in the following ways:

* Ignore (INDIFFERENT) Always Join

Default. If a transaction is active, the scope joins it. If not, the scope does not create a transaction.

* Always Begin (ALWAYS\_BEGIN)

A new transaction is started every time the scope is executed.

* Begin or Join (BEGIN\_OR\_JOIN)

Relevant only when execution order might vary (for example, due to asynchronous actions occurring outside the flow). If current flow processing has already started a transaction, the scope joins it. If not, the scope initiates a new transaction.

Until-Successful:

- The content of element 'until-successful' is not complete. Will get if until successful is not configured with single processor inside.

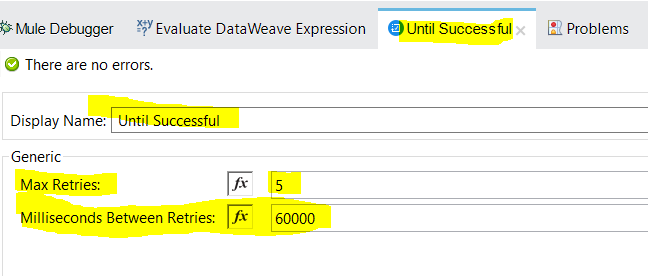
- The Until Successful scope executes processors within it, in order, until they all succeed or the scope exhausts the maximum number of retries.

-Until Successful runs synchronously.

- If any processor within the scope fails to connect or to produce a successful result, Until Successful retries all the processors within it, including the one that failed, 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.

-Routing is successful if no exception is raised or if the response matches an expression.



| **Field** | **XML** | **Description** |
| --- | --- | --- |
| Max Retries | maxRetries | Specifies the maximum number of retries that are allowed. Default value is 5.This attribute can be either a number or an expression that resolves to a number. An error message looks like this: Message: 'until-successful' retries exhausted. The Mule error type is MULE:RETRY\_EXHAUSTED. |
| Milliseconds Between Retries | millisBetweenRetries | Specifies, in milliseconds, the minimum interval between two retries. The actual interval depends on the previous execution, but it should not exceed twice this number. The default value is 60000 milliseconds (one minute). This attribute can be either a number or an expression that resolves to a number. |



-In the above screenshot,

When flow starts it will set the var array,set payload n prints the before until-successful logger once trigger comes on until successful(configured max-retries=5 n time between retry=5000)

since we have configured wrong creds for sfdc connector it will fail so since it is in until-successful scope

It will retry 5 times that i.e total execution of sfdc query operation is 1+5 times..

Retrying execution of event, attempt 1 of 5. Then same error

Retrying execution of event, attempt 2 of 5. Then same error

Retrying execution of event, attempt 3 of 5. Then same error

Retrying execution of event, attempt 4 of 5. Then same error

Retrying execution of event, attempt 5 of 5. Then same error

Then after exhausting all configured 5 retries it throws

Retry attempts exhausted. Failing... MULE:RETRY\_EXHAUSTED which is caught by on-error-continue @flow level then it executes **Query Execution failed logger**

**Behavior of the example application:**

* If the SFDC Query operation fails

The Until Successful scope retries the operation every 5000 milliseconds until the operation succeeds, with a limit of 5 retries. If the last execution fails, the scope throws a MULE:RETRY\_EXHAUSTED error. Then, the <on-error-continue> handles the error and executes the Logger with the message: Query Execution failed.

Before throwing MULE:RETRY\_EXHAUSTED, Mule also logs each unsuccessful attempt to retry execution of the processors on the Mule event, for example:

ERROR 2022-12-09 17:41:44,910 ... event: cdad31c0-782b-11ed-af21-147ddaaf4f97]

...UntilSuccessfulRouter: Retrying execution of event, attempt 1 of 5.

...

ERROR 2022-12-09 17:41:55,079 ... event: cdad31c0-782b-11ed-af21-147ddaaf4f97]

...UntilSuccessfulRouter: Retrying execution of event, attempt 5 of 5.

* If the sfdc query operation executes successfully in any of the attempts

The next processor after the sfdc query executes

- Every execution of the Until Successful scope starts with the same variables and values present before the execution of the Until Successful scope block. New variables or modifications to already-existing variables while processing one element are not visible in the next execution (in case there is an error). If the execution finishes correctly, the variables (and payload) are propagated to the rest of the flow.

Async Scope:

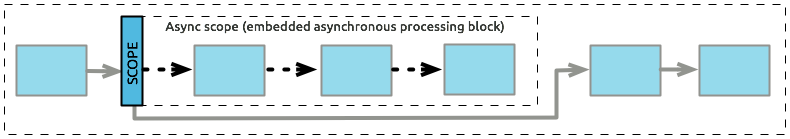
- The content of element 'async' is not complete.Will get if scope is not configured with any single processor inside.

-The Async scope is a branch processing block that executes simultaneously with the main flow.

- The main flow continues to execute while it initiates and processes the Async scope.

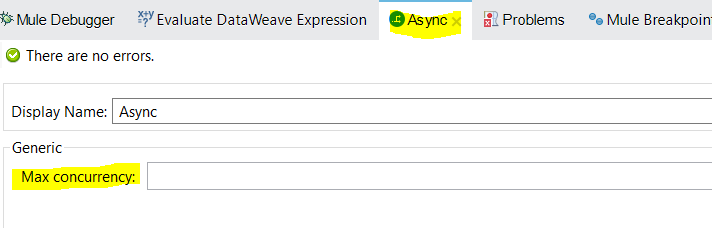
- The flow does not have to pause until the last processor in the asynchronous scope has completed its task.

-Async can be useful for executing time-consuming operations that do not require you to send a response back to the initiating flow (such as printing a file or connecting to a mail server).

-To facilitate simultaneous branch processing, the Async scope sends one copy of the message it has received to the first embedded message processor in its own processing block. At the same time, it sends another copy of the message to the next message processor in the main flow. 

Because the Async scope is executed in a "fire and forget" manner, the result of the processing within the scope is not available in the main flow.

| Field | Description |
| --- | --- |
| Display Name (name) | Name for the Async scope. |
| Max Concurrency (maxConcurrency) | Optional. Sets the maximum number of concurrent messages that the scope can process. By default, the container thread pool determines the maximum number of threads to use to optimize the performance when processing messages. When the scope is processing the maximum number of concurrent messages, it cannot receive additional requests.  Set maxConcurrency to 1 to cause the scope to process requests one at a time. |



**-Use Try scope to handle errors @async scope**, this is because the async scope does not inherit the error strategy from the flow level(see below screenshot)



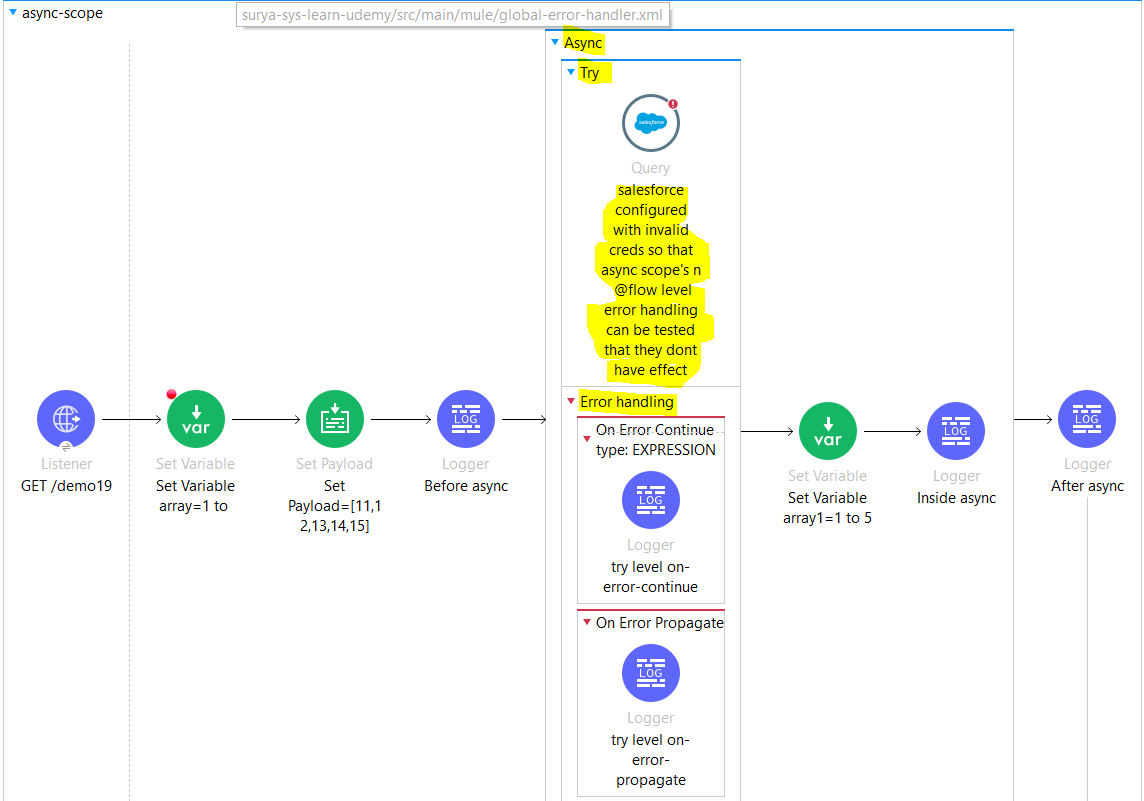
In the above case when flow starts,it executes set var array ,set payload n prints before async logger

After that the same mule message is copied n passed to 2 places for processing simultaneously

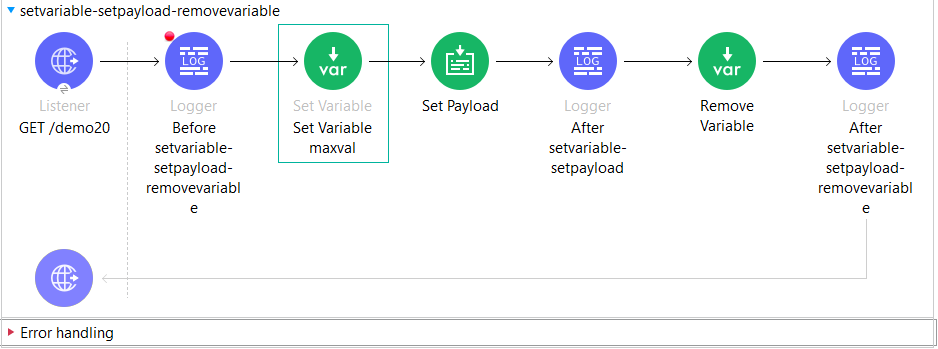
1)Async scope

2)outside of async scope i.e After async logger

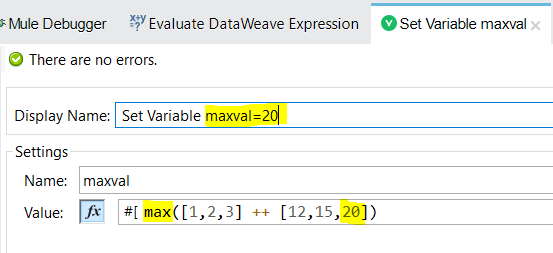
So since both async n outside processors of async are processing simultaneously so in above case it prints after sync logger first n sends the payload with 200 ok status as response to http client then goes to sfdc query executor but since the login details are not configured correctly so it will fail but it wont go to @flow level error handler so to handle the error we can use Try scope inside async scope to handle error as async does not go into flow level error handler…like below



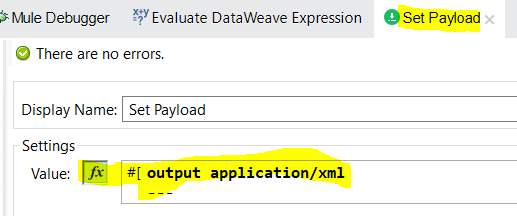
Set Variable,Set Payload,Remove Variable:



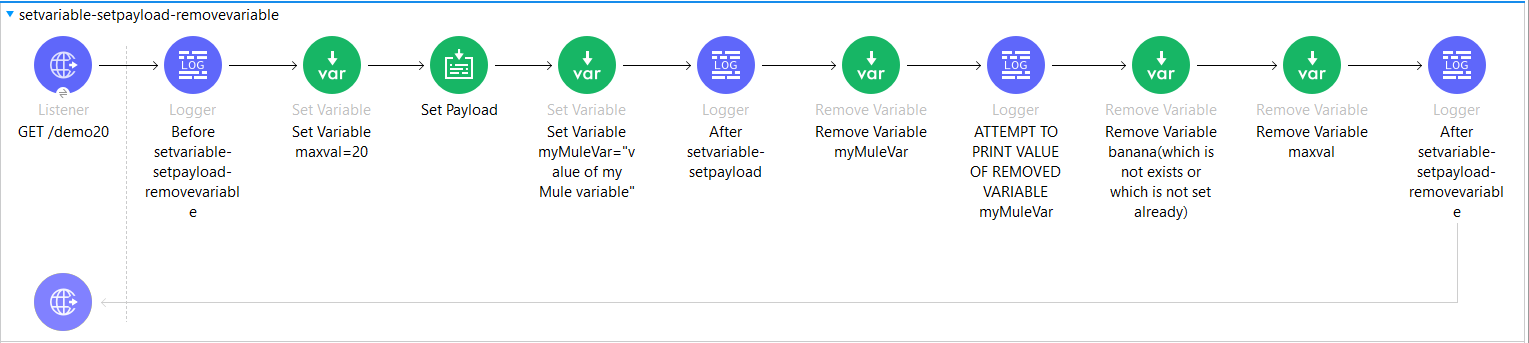
-Set variable is used to creates or updates a variable for use later in the flow. Variables are stored in a global variable object called 'vars' - for example, the variable 'myVar' would be accessed as 'vars.myVar' in an expression



-set payload used to sets the payload with the provided value or lets you update the payload of the message. The payload can be a literal string or a DataWeave expression.



-Remove Variable takes a Mule variable name and removes the variable from a Mule event.if the variable does not exist in the flow, the component logs a warning.if we attempt to access the value of a variable that does not exist or it is already removed it return a null value.



INFO 2022-12-15 08:33:20,155 ...event: 2f65e920-7c96-11ed-97ec-147ddaaf4f97]

ATTEMPT TO PRINT VALUE OF REMOVED VARIABLE: null

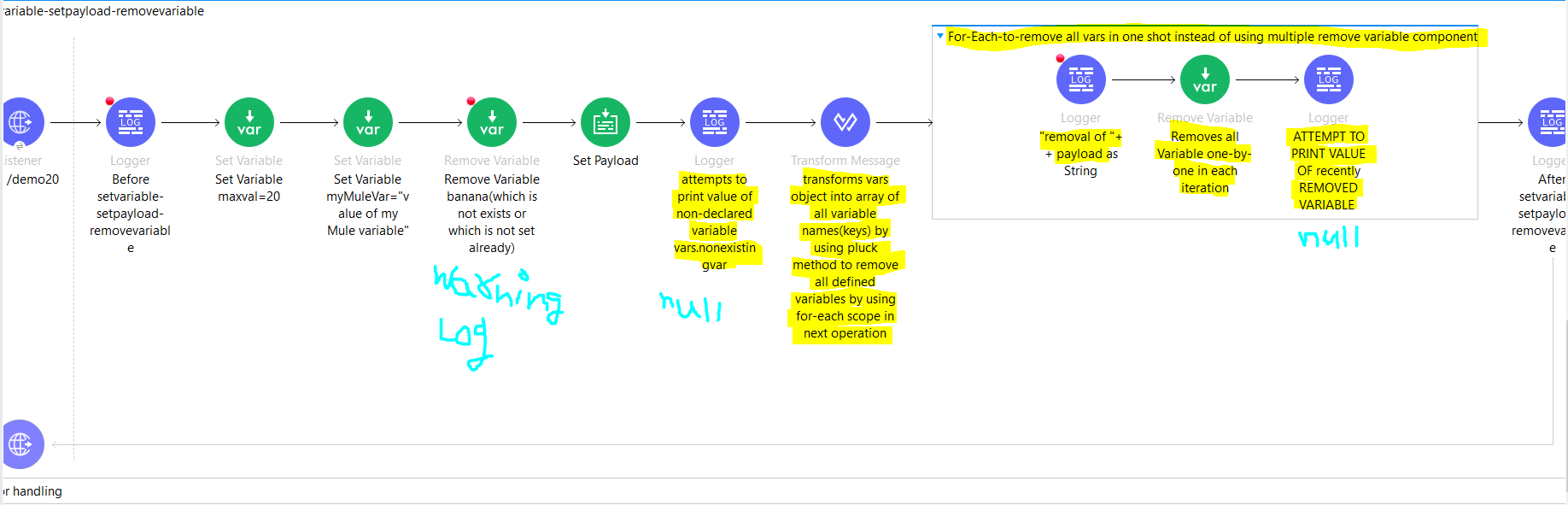
WARN 2022-12-15 08:33:20,156 ...event: 2f65e920-7c96-11ed-97ec-147ddaaf4f97]

...RemoveFlowVariableProcessor: There is no variable named 'banana'.

Check the 'variableName' parameter in the 'remove-variable' component at

remove-variable-ex/processors/4

**-In the below screenshot,we are trying to remove all variables by using one remove variable in foreach**



Execution behaviour goes like below:

1)LoggerMessageProcessor: {

"flowname": "setvariable-setpayload-removevariable",

"position": "Before setvariable-setpayload-removevariable",

"payload": "",

"attributes": {

"name": "Edna",

"age": "29",

"company": "Deloitte"

},

"vars": {}

}

2) WARN 2023-08-29 12:01:36,678 [[MuleRuntime].uber.98: [surya-sys-learn-udemy].setvariable-setpayload-removevariable.CPU\_LITE @532eda9e] [processor: setvariable-setpayload-removevariable/processors/3;

event: b0d31aa1-4635-11ee-90a1-0a6ac5085ff7] org.mule.runtime.core.internal.processor.simple.RemoveFlowVariableProcessor: There is no variable named 'banana'. Check the 'variableName' parameter in the

'remove-variable' component at setvariable-setpayload-removevariable/processors/3

3)LoggerMessageProcessor: attempts to print value of non-declared variable vars.nonexistingvar= null

**For each starts:**

4)LoggerMessageProcessor: "removal of variable name= maxval"

LoggerMessageProcessor: "ATTEMPT TO PRINT VALUE OF recently REMOVED VARIABLE= maxval and the val

ue which is returned is= null"

LoggerMessageProcessor: "removal of variable name= myMuleVar"

LoggerMessageProcessor: "ATTEMPT TO PRINT VALUE OF recently REMOVED VARIABLE= myMuleVar and the

value which is returned is= null"

**For each Completes:**

LoggerMessageProcessor: {

"flowname": "setvariable-setpayload-removevariable",

"position": "After setvariable-setpayload-removevariable",

"payload": [

"maxval",

"myMuleVar"

],

"attributes": {

"name": "Edna",

"age": "29",

"company": "Deloitte"

},

"vars": {}

}

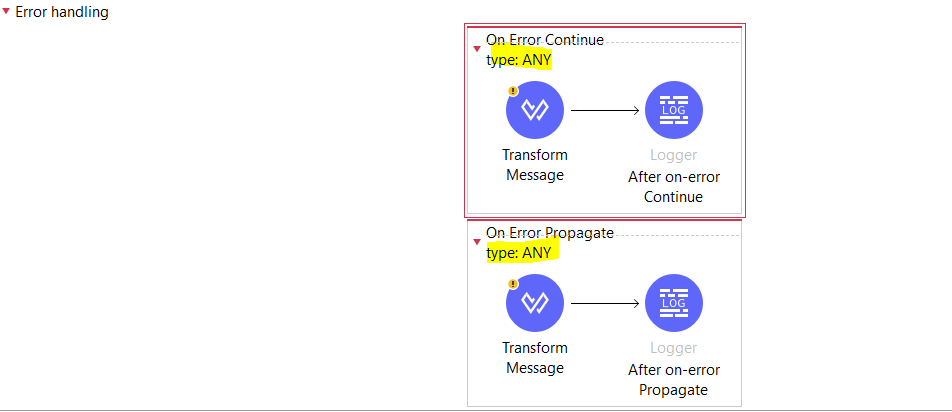
Error-handling:

-on-error propagate is a default behavior if any error got raised….

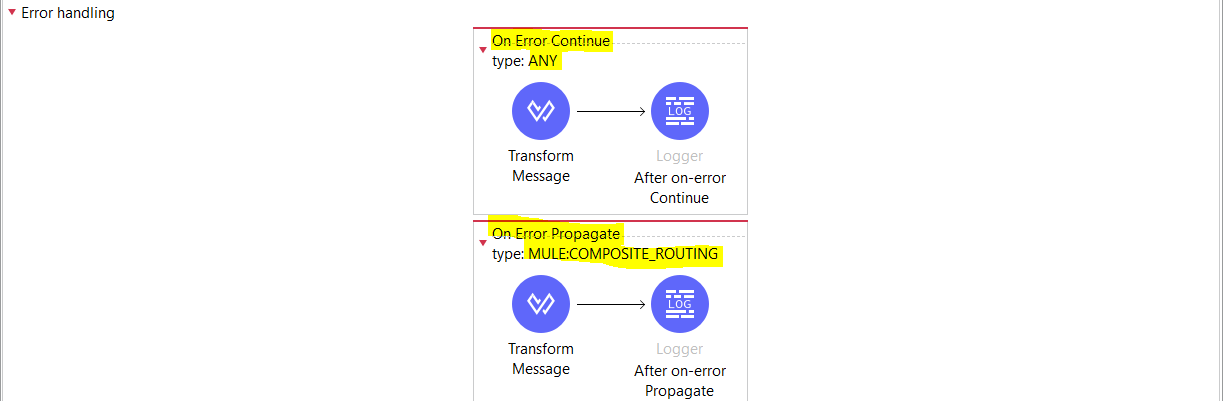
-on-error propagate component ,after executing other components declared in it, treats an error occured as error and sends the error response configured at listener end for eg: by default In HTTP listener(in Responses tab) for success we have payload in body with 200 ok status code n for error we have error.description in body with 500 server error status code configured as error response . so if we are using on-error propagate then we get error.description at listener end with 500 server error

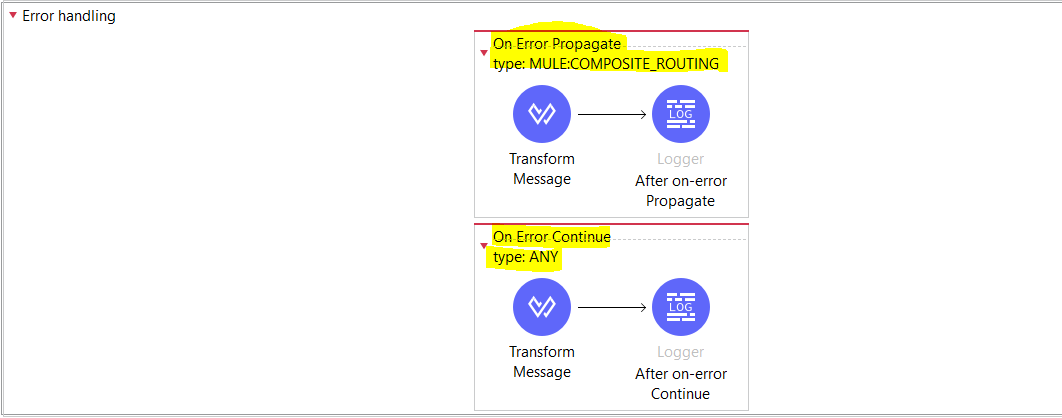
- on-error continue allows you to continue after executing other components.this component ,after executing other components declared in it, treats an error occured as success and sends the success response configured at listener end for eg: by default In HTTP listener(in Responses tab) for success we have payload in body with 200 ok status code n for error we have error.description in body with 500 server error status code configured as error response . so if we are using on-error continue then we get payload at listener end with 200 ok.

-so let say in a flow we have both on-error continue n on-error propagate error handlers defined with similarType (assume ANY) so whatever will be the first it will go into that n executes the components n treats as error or success accordingly.

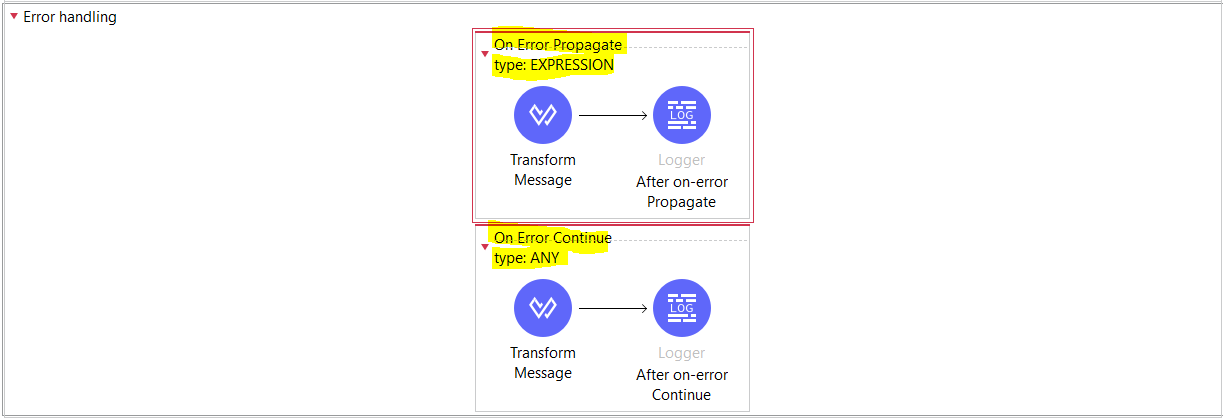


-so let say we have kind of setup like this where we have on-error continue on type ANY and on-error propagate on type MULE:COMPOSITE\_ROUTING and fyi the error which thrown is of type MULE:COMPOSITE\_ROUTING so in this case it will avoid on-error propagate on type MULE:COMPOSITE\_ROUTING as it is declared 2nd in order and type ANY is principle type so it will go into on-error continue on type ANY flow.

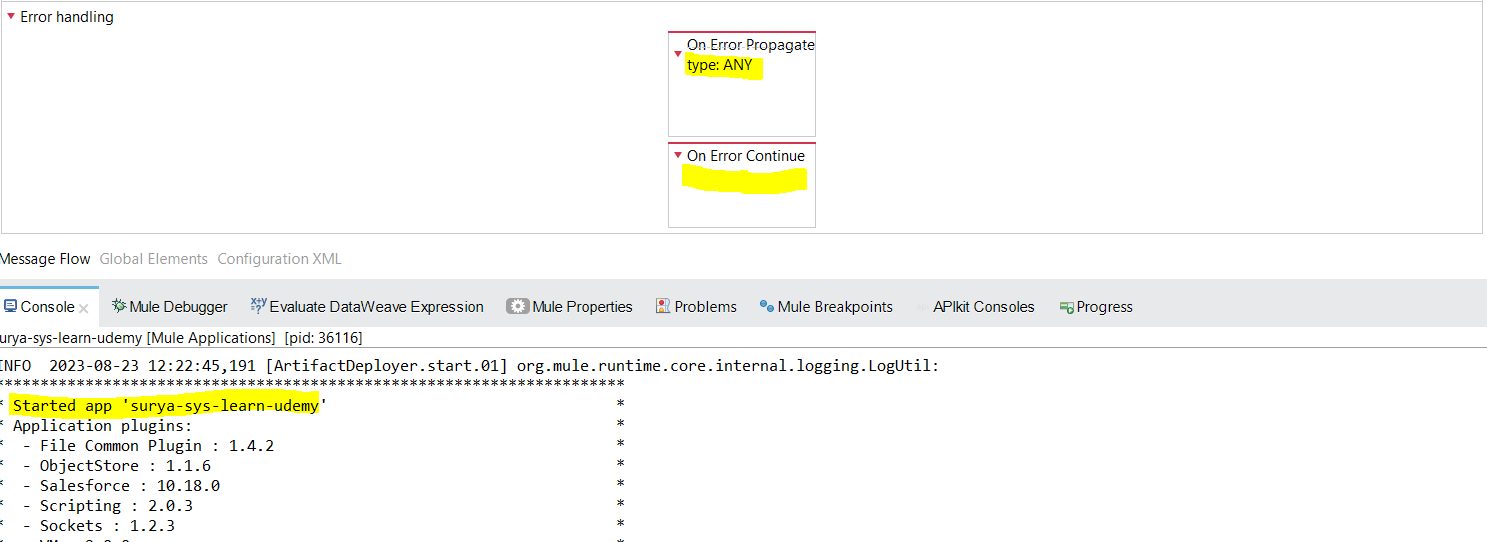


-in below case, let say error MULE:COMPOSITE\_ROUTING thrown so it will consider on-error propagate with type: MULE:COMPOSITE\_ROUTING as it is 1st in order n has the error type which is thrown

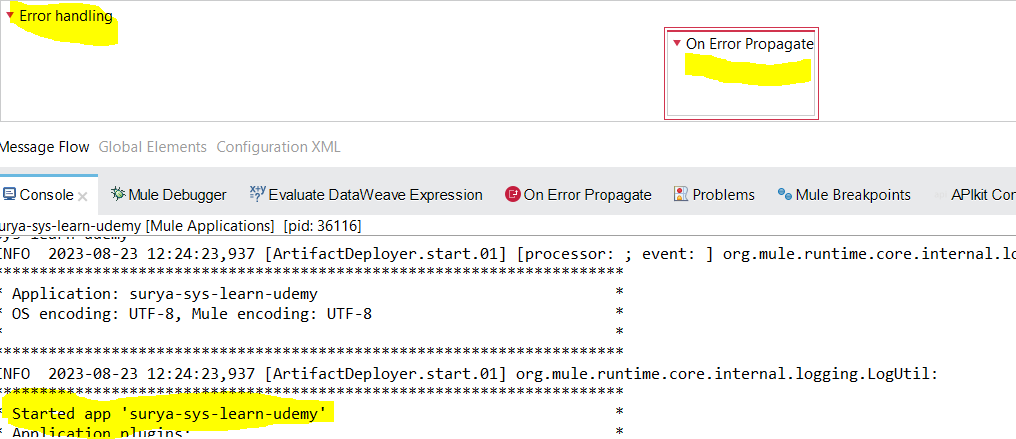
-in below case, let say error MULE:COMPOSITE\_ROUTING thrown so it will consider on-error continue with type: ANY this is becoz the 1st declared error handler on-error propagate is of type: EXPRESSION n the error thown is of type MULE:COMPOSITE\_ROUTING so it is not matching so it will go to next in the order there it finds on-error continue with type:ANY (which is principle(catches every type of error)) so in this case on-error continue with type: ANY will be executed…



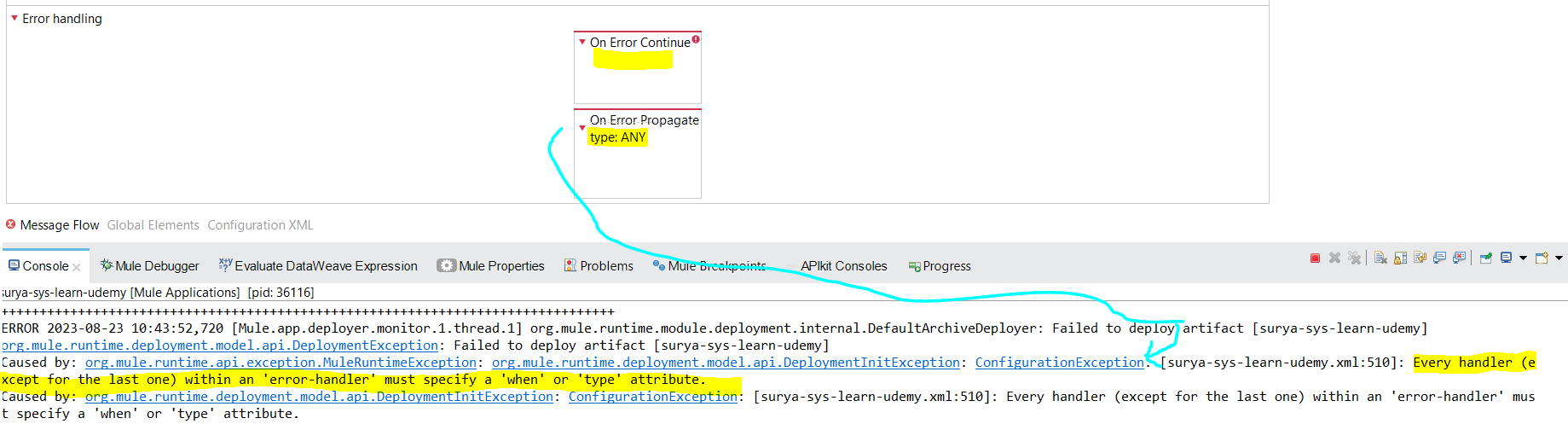
-Except the last one error-handler(on-error continue or on-erro propagate) defined in error handling section all rest error handlers should define either Type or When attribute,if we don’t have then our project will failed to deploy n gives Every handler (except for the last one) within an 'error-handler' must specify a 'when' or 'type' attribute.



-Incase we have only one error-handler(on-error continue or on-erro propagate) defined in error handling section then it will get successfully deployed n by default it will catch ANY errors



-Incase we have last error handler defined type but the others are not defined with when or Type attribute then also we will get above error…



Error-handler placements:-

-If no error handler is there then in case of error raised then it will consider default error handler that is 500 with error.description in body

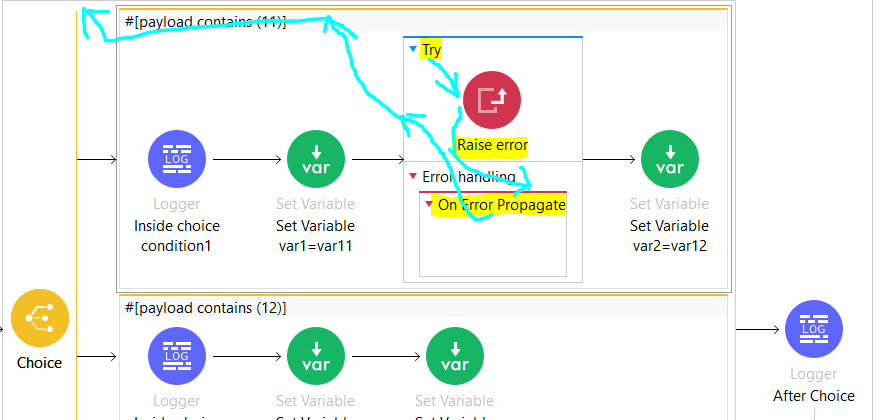
**@flow level:**

* 1. Only On-error-continue defined at flow level with matching type or when attribute for which error got raised so it will be catched by on-error-continue n then it will treat tht error as success n send payload n 200 ok status code in response
  2. Only On-error-propagate defined at flow level with matching type or when attribute for which error got raised so it will be catched by on-error-propagate n then it will treat tht error as error n send error.description n 500 internal server status code in response
  3. When we have both error-handlers defined at flow level with both the same matching type or when attribute( On-error-continue type =any , On-error-propagate type=any)for which error got raised then it will be catched by the first in ordered error handler defined at flow levelin this case it will trigger On-error-continue flow… but let say the error got raised type=MULE:COMPOSITE\_ROUTING and ( On-error-continue type =MULE:VALIDATION , On-error-propagate type=any) so in this case it will trigger On-error-propagate flow becoz ANY can catch any type of error

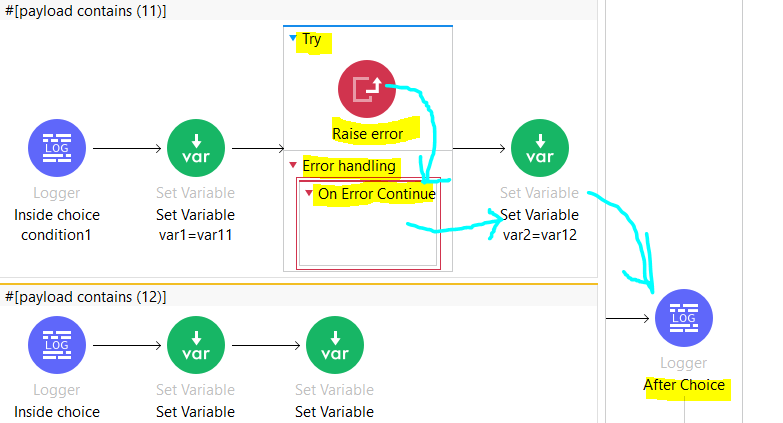
**@Component level:**

-If no error handler is there then in case of error raised then it will consider default error handler that is 500 with error.description in body

- if on-error propagate within TRY scopes error handling section then the error occurred will be propagated to the upper level n it will not continue to process the next processors i.e it will not execute var2=12 n after choice n will send error.description with 500 status code



-if on-error continue within TRY scopes error handling section then the error occurred will be ignored n it will continue to process the next processors i.e var2=12 n after choice logger will be printed



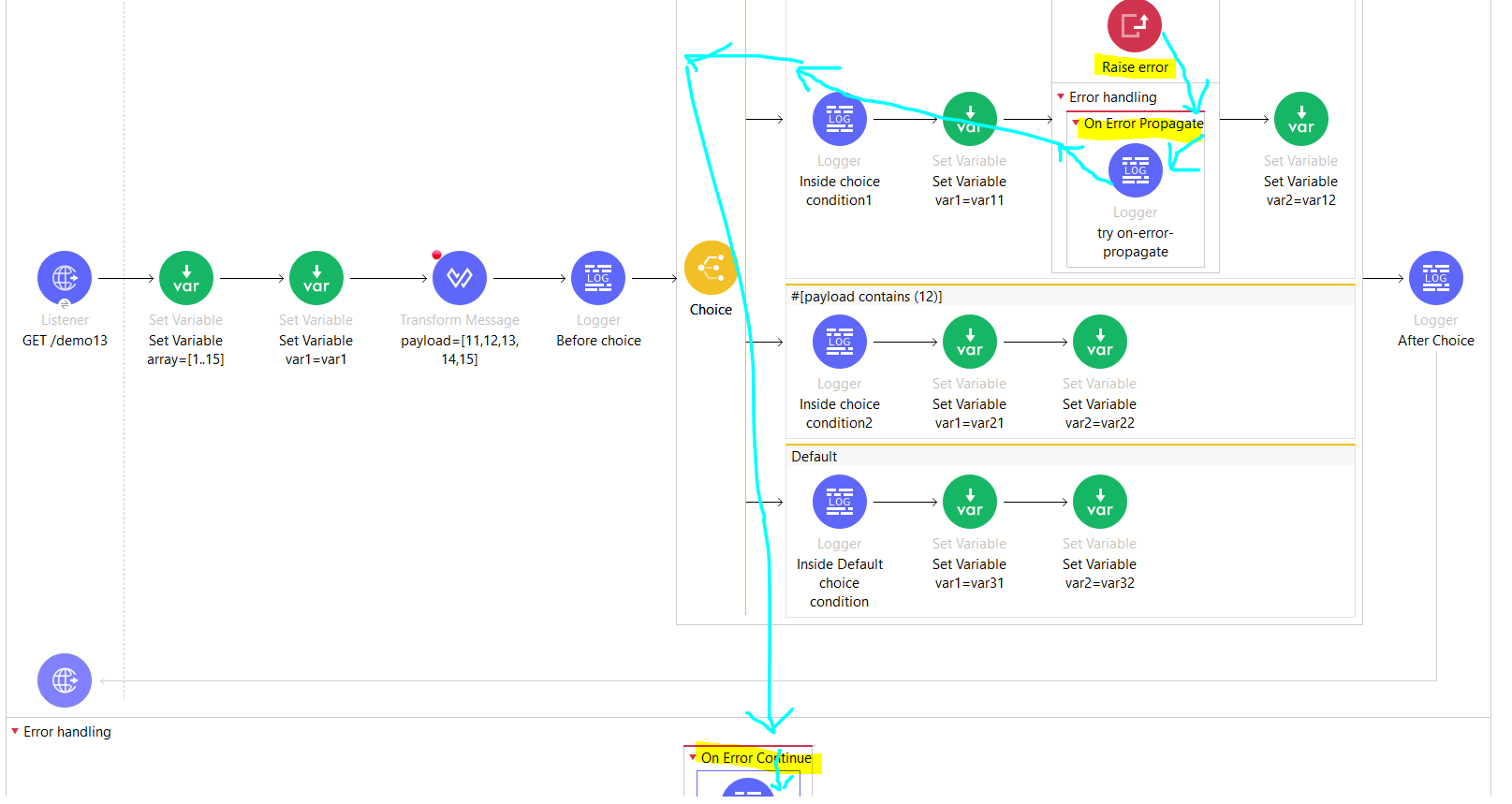
**@flow level + @Component level:**

--If no error handler is there then in case of error raised then it will consider default error handler that is 500 with error.description in body

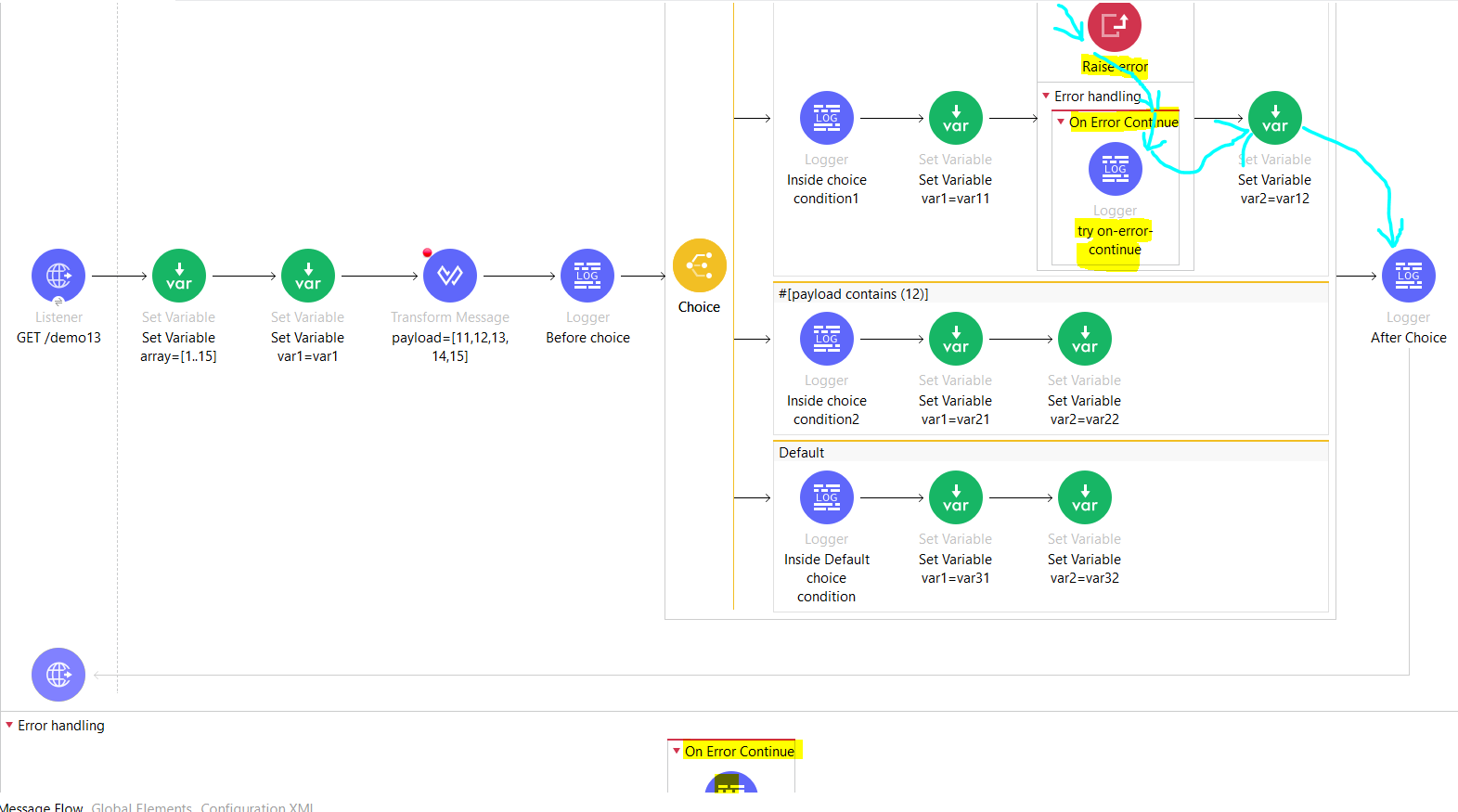
- if on-error propagate within TRY scopes @component level and on-error propagate within@flow level error handling section then the error occurred at raise-error level will be propagated to try scope on-error propagate and try on-error-propagate logger will execute after that that error will be propagated to flow level on-error propagate n flow on-error-propagate logger will get printed n finally will send error.description with 500 internal server error status code in body….



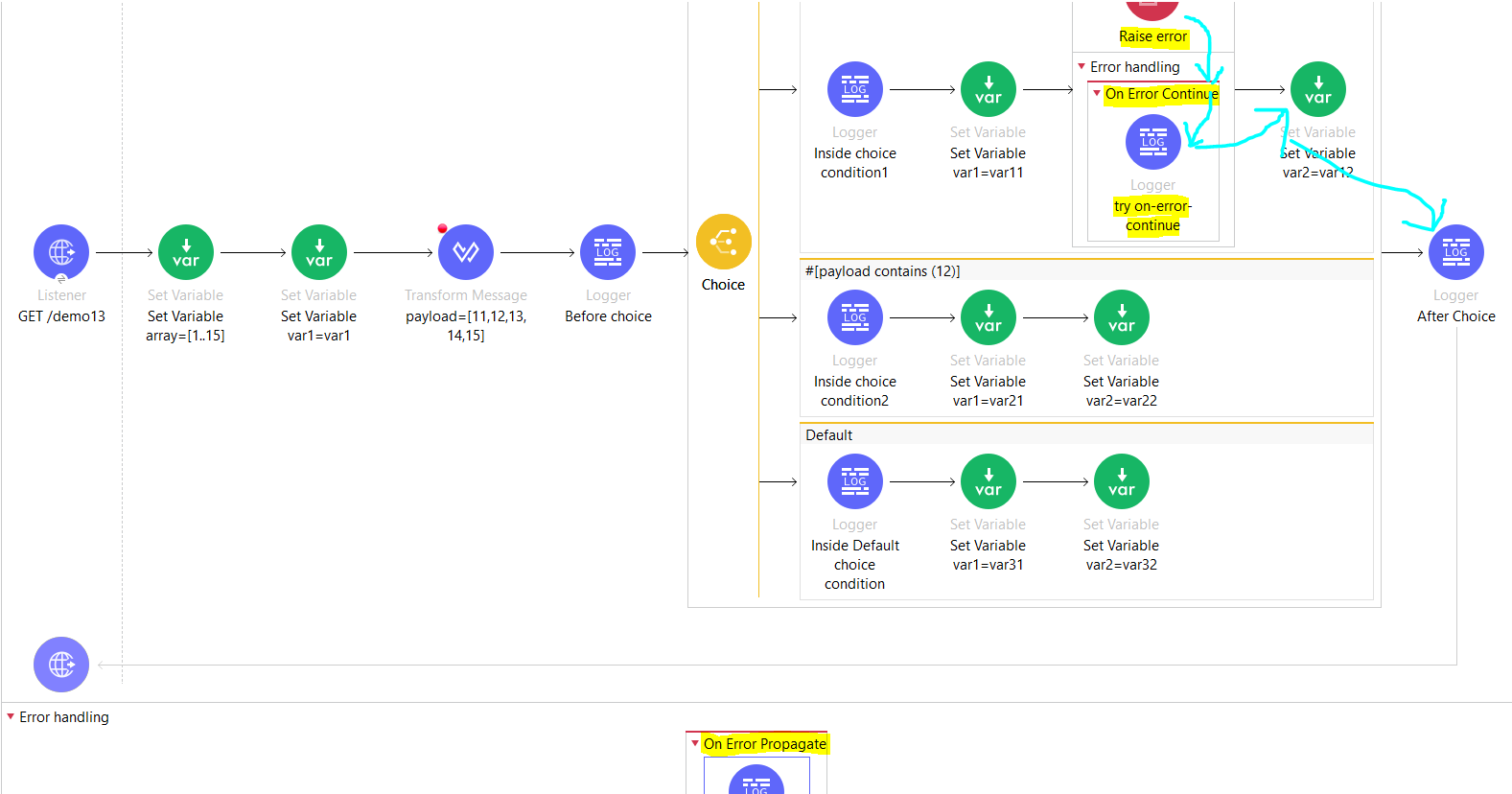
- if on-error propagate within TRY scopes @component level and on-error continue within@flow level error handling section then the error occurred at raise-error level will be propagated to try scope on-error propagate and try on-error-propagate logger will execute after that that error will be propagated to flow level on-error continue n flow on-error-continue logger will get printed n finally will send payload with 200 ok status code in body….



- if on-error continue within TRY scopes @component level and on-error continue within@flow level error handling section then the error occurred at raise-error level will be handled at try scope on-error continue n then it will execute try on-error continue logger n will go to var2=12 n after choice n will send payload with 200 status code in body….



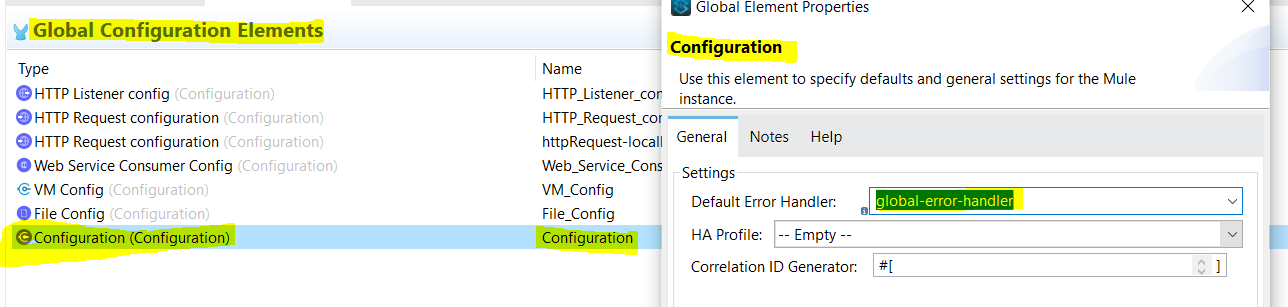
- if on-error continue within TRY scopes @component level and on-error propagate within@flow level error handling section then the error occurred at raise-error level will be handled at try scope on-error continue n then it will execute try on-error continue logger n will go to var2=12 n after choice n will send payload with 200 status code in body….



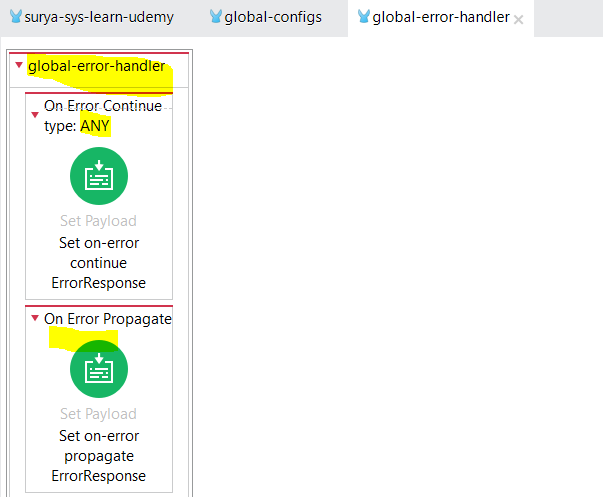
**@Global level**

-If no error handler is there then in case of error raised then it will consider default error handler that is 500 with error.description in body

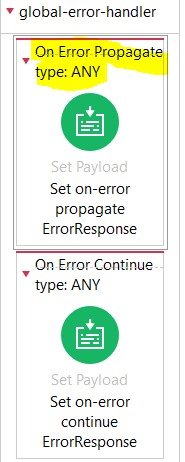
-To Configure global-error-handler, we need to specify default error handler as global-error-handler file name in configuration under Global-Configuration-Elements tab



-if only global-error-handler is configured (not configured error-handler at flow or component level) then when error raised it will go to matching type or when attribute n execute or else it will execute default error-handlr that is error.description with 500 status code.In below case it will go to on-error-continue n send payload with 200 ok response.



In below case it will go to on-error-propagate n send error.description with 500 internal server error response.



**-if global-error-handler=on-error propagate,component= on-error propagate**

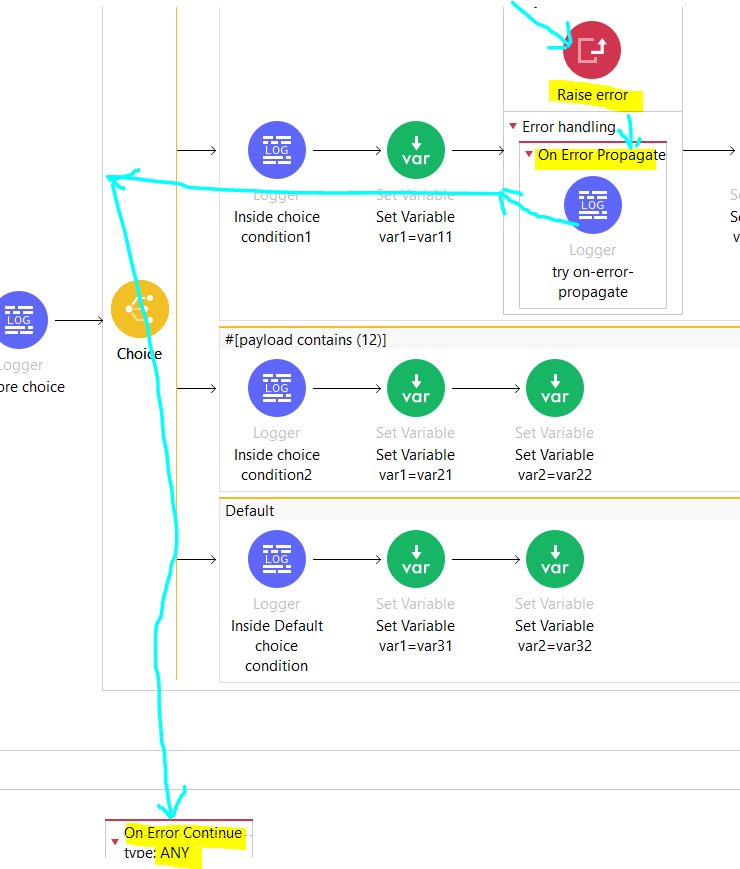
it will go to on-error-propagate of both component n global-error-handler n send error.description with 500 internal server error response.

**if global-error-handler=on-error propagate,component= on-error propagate,flow= on-error propagate**

it will go to on-error-propagate of both component n flow n send error.description with 500 internal server error response.in this case it will not go into the global-error handler

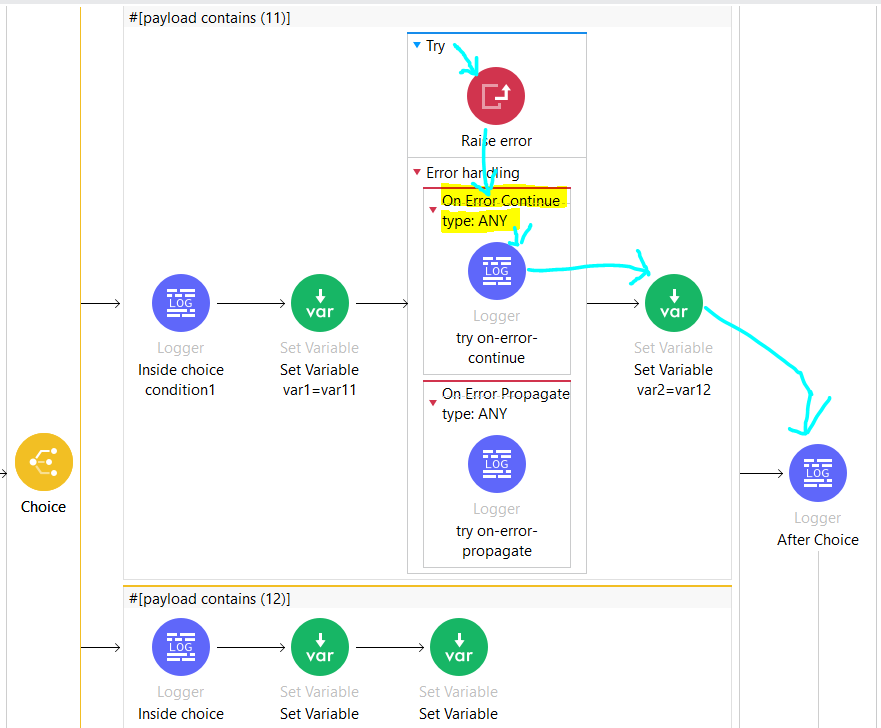
**if global-error-handler=on-error propagate,component= on-error propagate,flow= on-error continue**

it will go to on-error-propagate of component print the log n then flow level on-error continue will get triggered n print the log n send payload with 200 ok response in the body.in this case it will not go into the global-error handler also



**-if global-error-handler=on-error propagate,component= on-error continue,flow= on-error continue**

it will go to on-error-continue of component print the log n then flow continues n executes var2=12 n after choice logger n finally sends payload with 200 ok in this case trigger wont go in @flow n @global level error handler



**-for below case,we have two error scenarios where 1 is handled with try scopes error handling on-error continue so it will execute var2=12 n after choice n then 2nd error raised which has no error handler @component so it will go inside @flow level where we have on-error continue so it will send payload with 200 ok status in body**



**How To deploy the app in Cloudhub:**

* **Through** [**Anypoint Runtime Manager**](https://docs.mulesoft.com/cloudhub-1/deploying-to-cloudhub#deploy-an-application-from-runtime-manager)**(Manual Process)**

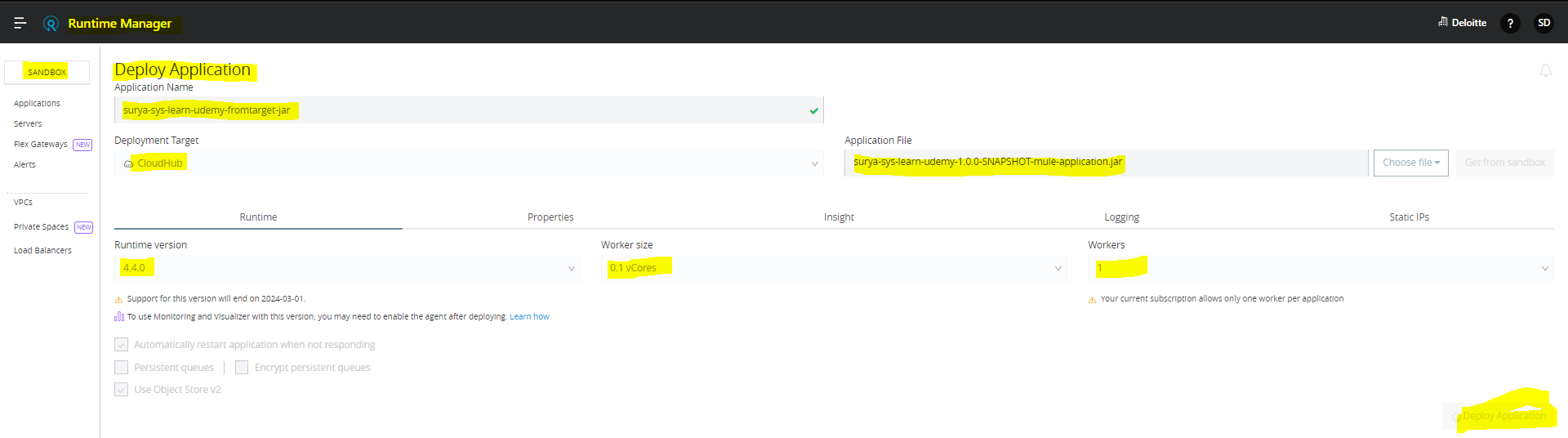
1)Sign in to Anypoint Platform.

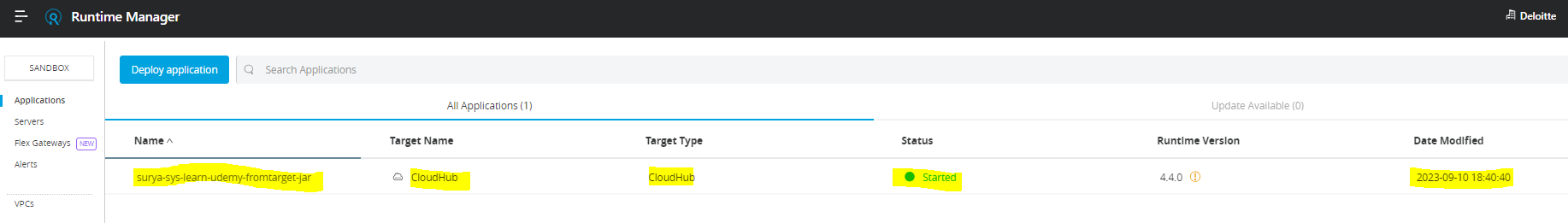
2)Select Runtime Manager,choose environment accordingly to which you want to deploy

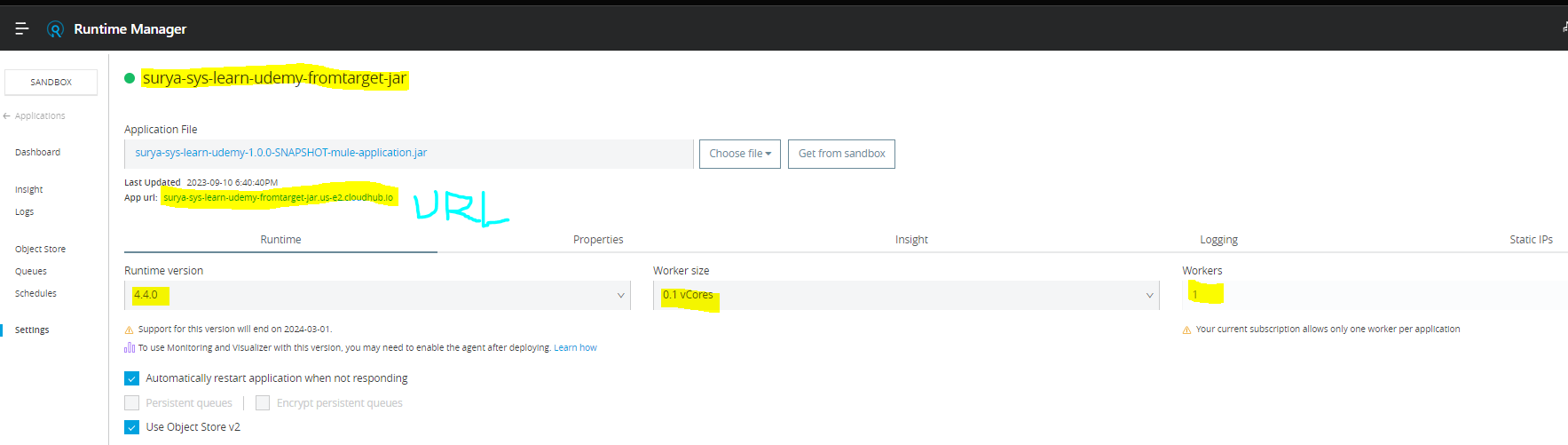
3)In the Applications page, click Deploy application.

u will get page like below

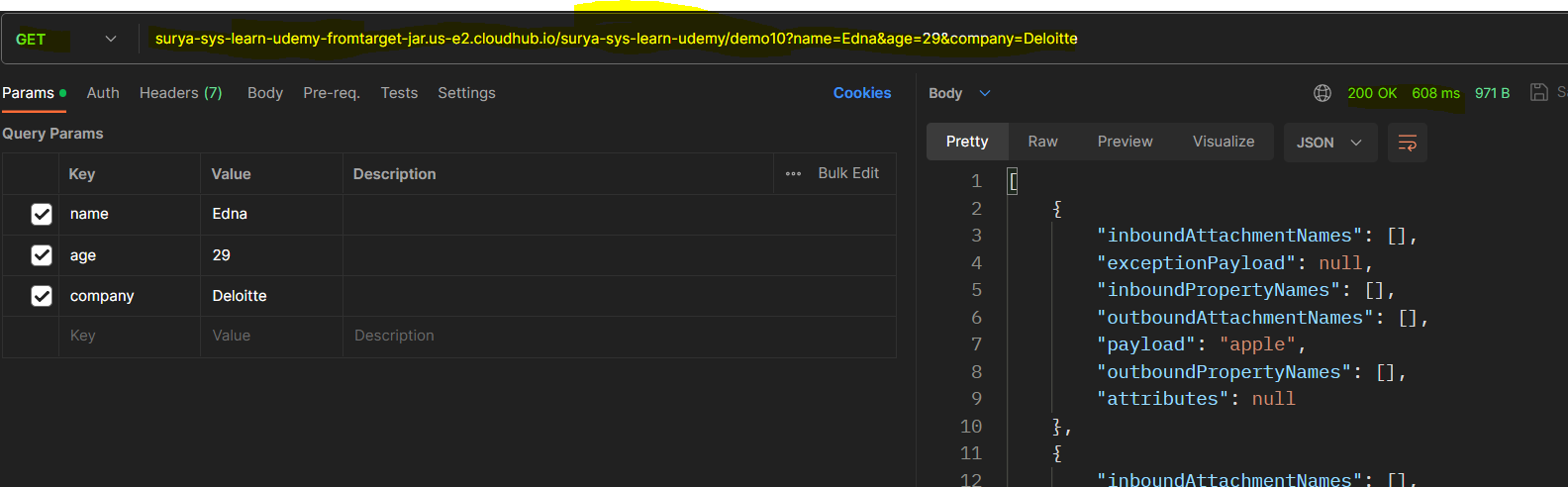
where you can give name to application,mention deployment target,choose file from either exchange or local .zip or jar file or else you can copy the already deployed project from other env to selected env, ,choose runtime version ,mention worker n worker size,u can add runtime properties,logging,static ip,insights such as metadata of mule transaction



* 1. Once deployment of an app is done you will see the selected env has that app status as started

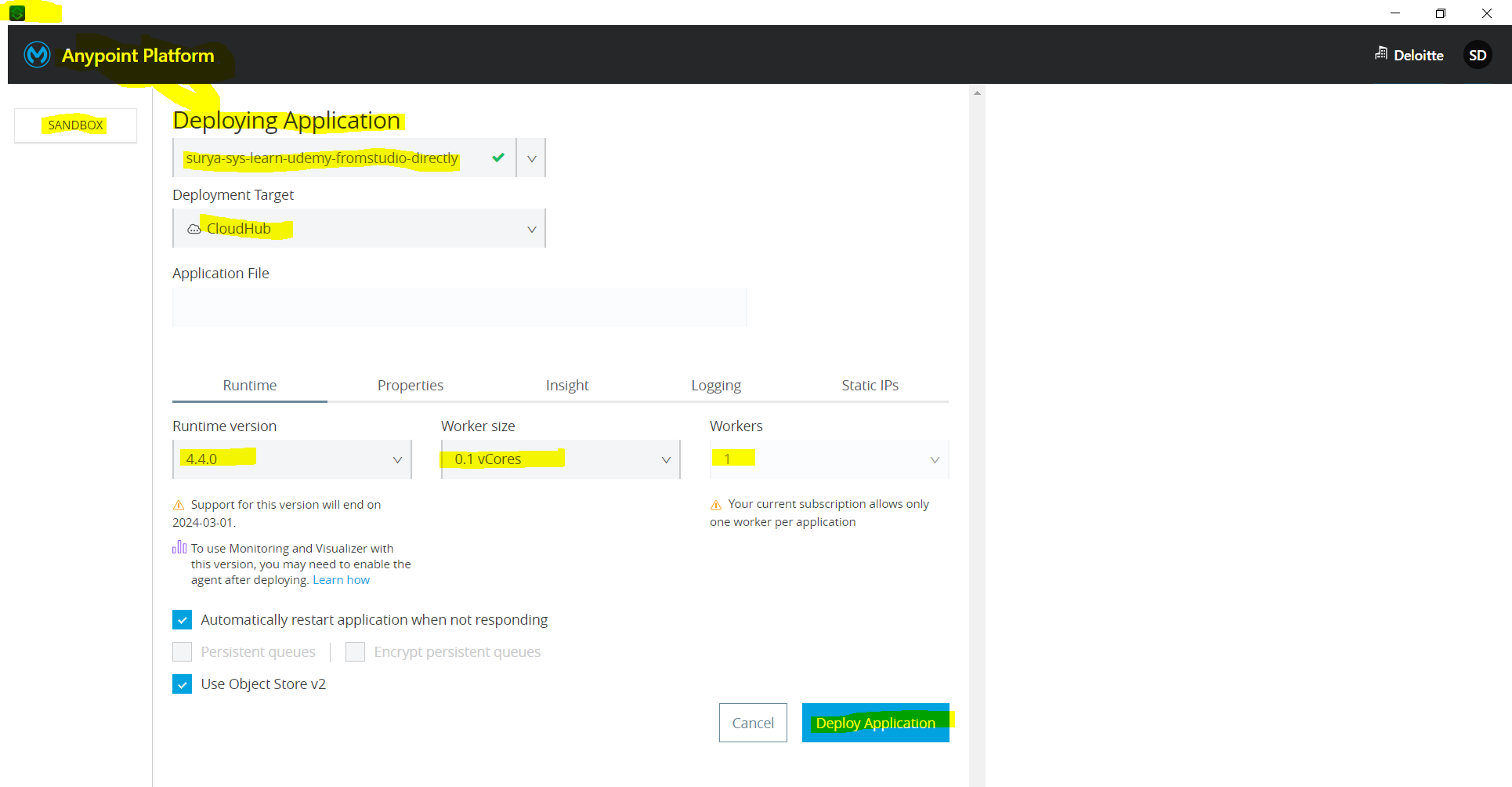


Instead of accessing the mule project locally To access the project directly from cloudhub ,we can use the mentioned App URL in the screenshot it will work .



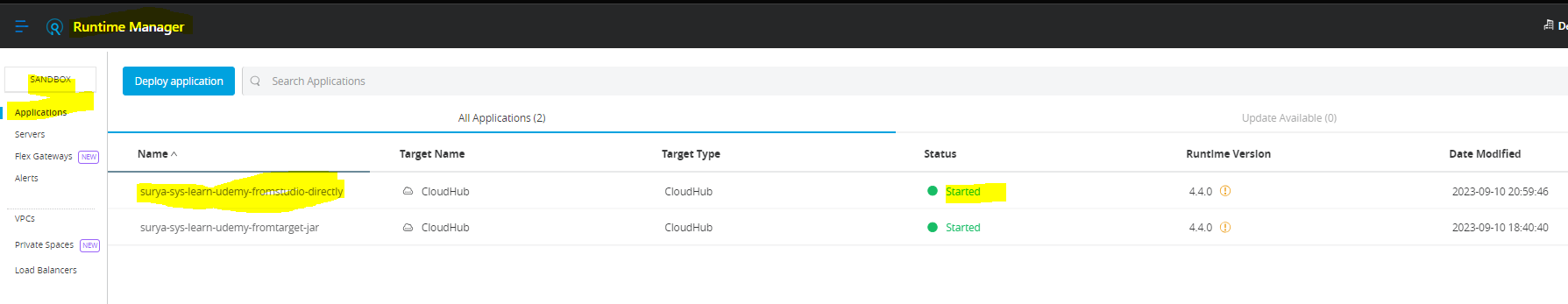
* **Through** [**Anypoint Studio**](https://docs.mulesoft.com/cloudhub-1/deploying-to-cloudhub#deploy-an-application-from-studio)

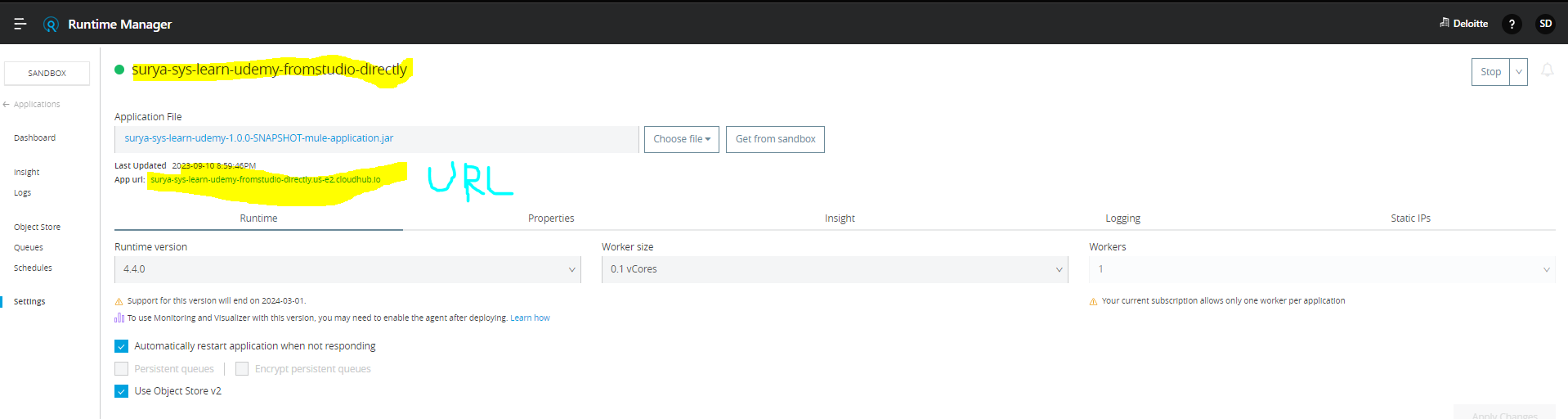
1. To deploy an app from anypoint studio,go to project in project explorer->right click->select anypoint platform->Deploy to cloudhub u will get below screen



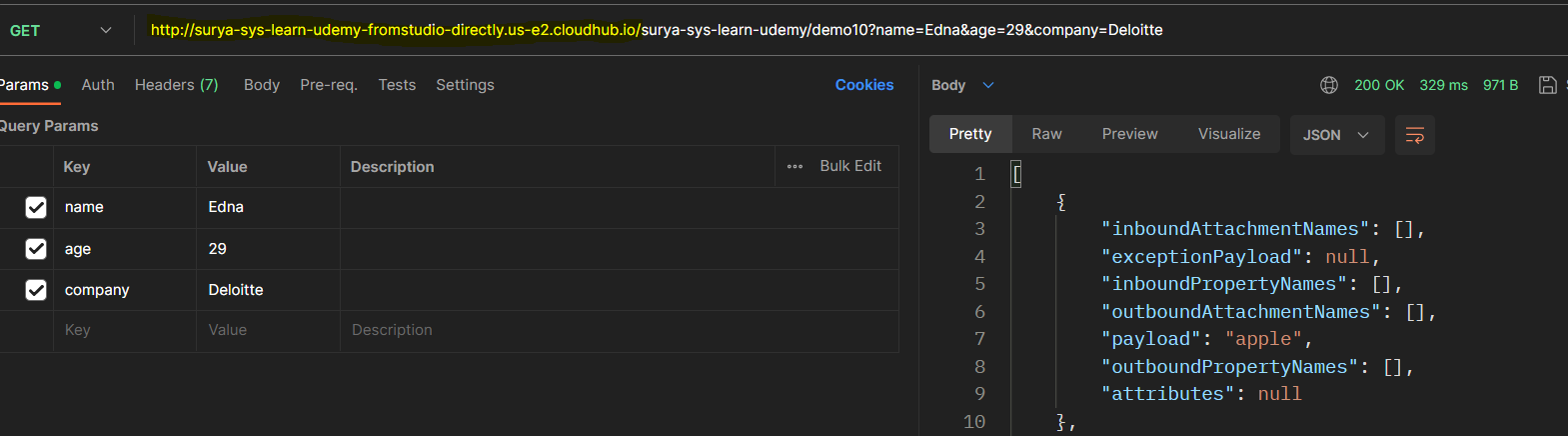
Select the env to deploy,give application name,select target,runtime,worker size ,workers ,define runtime properties etc n click on deploy application

2)once deployed ,u can login into anypoint platform->runtime manager->select env on which u deployed the app->u will be able to see the app listed like below





To access the app from cloudhub use the app url ,it works



**Below URL also will work without using http also**

**surya-sys-learn-udemy-fromstudio-directly.us-e2.cloudhub.io/surya-sys-learn-udemy/demo10?name=Edna&age=29&company=Deloitte**