**The point where Mule 4 changes are as follows:**  
• Mule Message  
• Expression Language  
• Connectors  
• Error Handlings  
• Dataweave  
• Studio 7  
• Runtime Engine  
  
**Mule Message Changes from Mule 3.x to Mule 4:-----**1. Inbound properties are now attributes.  
2. In Mule 3, connectors and transports that need to send additional data, such as headers, must explicitly specify Outbound properties. In Mule 4, you can set each of those separately using an individual DataWeave expression for each one of them, without introducing any side effects in the main flow  
3. Session properties not required.  
4. HTTP now relies on DataWeave’s multipart format support.  
5. Email and Web Service Consumer allow attachments to be explicitly added using DataWeave  
6. In Mule 3, components that returned multiple payloads, used a special structure called the MuleMessageCollection. In Mule 4, any component that needs to deal with multiple messages can simply set the payload of the message to a List of Mule Messages.  
  
**Expression Language Changes from Mule 3.x to Mule 4:-----**  
In Mule 3, you must learn both the Mule Expression Language (MEL) and DataWeave. MEL forces you to convert your payloads from binary data (such as XML or JSON documents) into Java objects so that you can write expressions that access that data, for example, when routing to a specific location.  
In Mule 4, DataWeave is the default expression language. Combined with the built-in streaming capabilities, this change simplifies many common tasks:  
• Events can be routed based on payload data, without requiring conversion to Java objects.  
• Binary data can easily be queried from an expression anywhere in your flow, for example, when logging.  
• Streaming now happens transparently. You no longer need to worry about larger-than-memory data streams or about consuming a stream twice.  
  
**Connectors Changes from Mule 3.x to Mule 4:-----**  
The Mule 4 distribution ships with no connectors or modules. Instead, each application includes the modules it needs. This approach has the following advantages:  
• Faster innovation in core modules/connectors because all connectors have their own release cycle. New features and bug fixes can become available without the need to wait for the next Mule Runtime release.  
• Ability for different apps to use different versions of the same module. This allows for easier adoption of new releases without the need to run a full regression on applications that don’t require the upgrade.  
• Consistent approach.  
  
**Error Handlings Changes from Mule 3.x to Mule 4:------**  
Mule 4 has redesigned error handling by introducing the error-handler component, which can contain any number of internal handlers and can route an error to the first one matching it. Such handlers are on-error-continue and on-error-propagate, which both support matching through an error type (or group of error types) or through an expression (for advanced use cases).   
If an error is raised in Mule 4, an error handler will be executed and the error will be routed to the first matching handler. At this point, the error is available for inspection, so the handlers can execute and act accordingly, relative to the component where they are used (a Flow or Try scope):  
• An on-error-continue will execute and use the result of the execution, as the result of its owner (as if the owner had actually completed the execution successfully). Any transactions at this point would be committed as well.  
• An on-error-propagate will roll back any transactions, execute, and use that result to re-throw the existing error, meaning its owner will be considered as “failing.”  
Mule 3 only allows error handling at the flow level, forcing you to extract logic to a flow in order to address errors. In Mule 4, we’ve introduced a Try scope that you can use within a flow to do error handling of just inner components. The scope also supports transactions, which replaces the old Transactional scope.   
Mule 4 now also allows for mapping default errors to custom ones. The Try scope is useful, but if you have several equal components and want to distinguish the errors of each one, using a Try on them can clutter your app. Instead, you can add error mappings to each component, meaning that all or certain kind of errors streaming from the component will be mapped to another error of your choosing. If, for example, you are aggregating results from 2 APIs using an HTTP request component for each, you might want to distinguish between the errors of API 1 and API 2, since by default, their errors will be the same.

|  |  |
| --- | --- |
| **Deployment Mechanism** | **Instructions** |
| CloudHub | CloudHub is Mule 4 ready. |
| Hybrid Deployment | You must install the standalone version of Mule 4 your servers. |
| Anypoint Platform Private Cloud Edition (PCE) | PCE does not currently support Mule 4. |
| Anypoint Platform for Pivotal Cloud Foundry (PCF) | PCF does not support Mule 4. |

**Migrating from DataWeave version 1 to 2**

1. **Dataweave Header Content**

|  |  |
| --- | --- |
| **DataWeave 1.0** | **DataWeave 2.0** |
| %dw 1.0 | %dw 2.0 |
| %output | output |
| %var | var |
| %function | fun |
| %namespace | ns |

## Flow Controls

## When Otherwise

The when otherwise statement is replaced by if else, for example:

Mule 3 Example: DataWeave 1

{ orderStatus: "complete" when flowVars.purchaseOrderStatus == "C" otherwise "incomplete"

}

Mule 4 Example: DataWeave 2

{ orderStatus: if(vars.purchaseOrderStatus == "C") "complete" else "incomplete"

}

## Pattern Matcher

Pattern matching changed in DataWeave 2. It adds the keyword case and else (instead of default). You also no longer separate cases with commas (,) since they are now explicitly separated by the case keyword.

Mule 3 Example: DataWeave 1

'world' match {

:string -> true,

default -> false

}

Mule 4 Example: DataWeave 2

'world' match {

case is String -> true

else -> false

}

## Type References

The : was removed from the type references and are now all camel case, so :string is now String

## Object Coercion

## In DataWeave 1.0, selecting a key-value pair from an object required you to do something like this:

Mule 3 Example: DataWeave 1

%var payload = {a: 1, b:2}

---

payload.a as :object

The DataWeave 1.0 expression above returns {a:1}. Because this is a coercion, it is also included in the auto-coercion mechanism and generates undesired or unexpected results.

In DataWeave 2.0, the coercion is removed, and a new selector (&) is introduced to select key-value pair parenthesis.

Mule 3 Example: DataWeave 2

var payload = {a: 1, b:2}

---

payload.&a

## Conditional Key-Value Parenthesis

In DataWeave 1.0, conditional key-value pairs are declared with the when keyword.

Mule 3 Example: DataWeave 1

{

(a: 1) when payload.product == "Mule"

}

In DataWeave 2.0, you use the if keyword.

Mule 4 Example: DataWeave 2

{

(a: 1) if payload.product == "Mule"

}

# **Migrating Core Components**

|  |  |
| --- | --- |
| **Mule 3.x Component** | **Mule 4.0 Replacement** |
| All router | Use scatter gather instead or execute the operations one after the other with different target attributes and then use DW to merge results. |
| async[‘processingStrategy’] | Async processing strategies are no longer needed with the non-blocking execution engine. Use an “async” scope wrapping all the components if you want them to behave asynchronously. |
| asynchronous-processing-strategy | Async processing strategies are no longer needed with the non-blocking execution engine. Use an “async” scope wrapping all the components if you want them to behave asynchronously. |
| bridge | Removed. Use flows. |
| catch-exception-strategy | Use error-handler with an on-error-continue strategy. |
| choice-exception-strategy | Use error-handler with different strategies inside using error type selection or when attribute. |
| combine-collections-transformer | No longer needed with the simplified message model. MuleMessageCollections are replaced with arrays of Mule Messages, which can be merged or iterated through using any Mule component, such as DataWeave or foreach. |
| component | Use the Java module. |
| composite-source | Create one flow per each source and invoke a private flow using flow-ref for the common functionality. |
| configuration[‘defaultExceptionStrategy-ref’] | New name is defaultErrorHandler-ref |
| configuration[‘flowEndingWithOneWayEndpointReturnsNull’] | No longer needed with new execution engine. |
| configuration[‘useExtendedTransformations’] | Removed. |
| configuration[enricherPropagatesSessionVariableChanges] | Removed. |
| copy-attachments | Use Transform component & DataWeave. |
| copy-properties | Use Transform component & DataWeave. |
| custom-agent | Removed. |
| custom-aggregator | Removed. |
| custom-connector | Use SDK to build a custom connector. |
| custom-exception-strategy | No longer needed. Use error-handlers. |
| custom-lifecycle-adapter-factory | No replacement. |
| custom-object-store | Use object store module extension to create custom stores. |
| custom-processor | Use DataWeave, scripting component, the Mule SDK, or the Java module. |
| custom-queue-store | Use the new VM Connector. |
| custom-router-resolver | Removed. |
| custom-router | Removed. |
| custom-service | Use flows. |
| custom-source | Use the Mule SDK to write a new event source. |
| custom-splitter | Use DataWeave or the Mule SDK. |
| custom-transaction-manager | Removed. |
| custom-transaction | Removed. |
| default-exception-strategy | Use error-handler with an on-error-propagate strategy. |
| default-in-memory-queue-store | Use the new VM Connector. |
| default-persistent-queue-store | Use the new VM Connector. |
| default-service-exception-strategy | Use error-handler with an on-error-propagate strategy. |
| endpoint | Use connectors. |
| exception-strategy | Use error-handlers. |
| expression-component | Use DataWeave, scripting component, the Mule SDK, or the Java module. |
| file-queue-store | Use the new VM Connector. |
| flow[‘processingStrategy’] | Removed. Non-blocking execution engine ensures that users do not need to tune this. |
| idempotent-message-filter | Replaced by idempotent-message-validator. |
| idempotent-redelivery-policy | New name is redelivery-policy. |
| idempotent-secure-hash-message-filter | Replaced by idempotent-message-validator |
| in-memory-store | Use object store module extension to create custom stores. |
| inbound-endpoint | Use connectors. |
| include-entry-point | Use the Java module. |
| interceptor-stack | Removed. Use custom policies. |
| invoke | Use the Java module. |
| jndi-transaction-manager | Removed. |
| jrun-transaction-manager | Removed. |
| legacy-abstract-exception-strategy | Use the new error handlers. |
| managed-store | Use object store module extension to create custom stores. |
| message-properties-transformer | Use Transform component and DataWeave. |
| model | Removed. Use flows instead. |
| mule[‘version’] |  |
| outbound-endpoint | Element moved from the core namespace to the new transports namespace. |
| parse-template |  |
| poll → watermark | This capability has been built inside event sources from many connectors, like Salesforce, Database, SFTP, File, etc. You can also use the object-store module to manually set the watermark values. |
| poll | Replaced with scheduler source. |
| pooling-profile[‘disabled’] | New attribute |
| processor | Use DataWeave, scripting component, the Mule SDK, or the Java module. |
| prototype-object | Use Java module or Spring module. |
| queue-profile | Removed. |
| queue-store | Removed. |
| recipient-list | Removed. |
| reconnect-custom-notifier | Removed. |
| reconnect-custom-notifier | Removed. |
| reconnect-custom-strategy | Removed. |
| reconnect-custom-strategy | Removed. |
| reconnect-notifier | Removed. |
| remove-attachment | No longer needed. Attachments can be stored as variables. |
| remove-property | Properties no longer exist in the new message model. You can store attributes from the Mule message in a variable. |
| response | Removed. No longer needed. |
| request-reply | Mule 4 will no longer support request-reply for all connectors. Connectors that had a “request-reply” behavior will provide a “request-reply” operation built in, such as the JMS consume operation. |
| resin-transaction-manager | Removed. |
| rollback-exception-strategy | Use error-handler with an on-error-propagate strategy. |
| scatter-gather[‘threading-profile’] | No longer needed now that Mule 4 is non-blocking. |
| seda-model | No more SEDA queues in Mule 4. The execution engine in Mule 4 is non-blocking. |
| service | Use flows. |
| set-attachment | No longer needed. Attachments can be stored as variables. |
| set-property | Properties no longer exist in the new message model. You can store attributes from the Mule message in a variable. |
| set-session-variable | Session variables have been removed. Users must explicitly pass headers across transport boundaries. |
| simple-in-memory-queue-store | Use the new VM Connector. |
| simple-service | Use flows. |
| simple-text-file-store | Use object store module extension to create custom stores. |
| singleton-object | Use Java module or Spring module. |
| spring-object | Use Java module or Spring module. |
| synchronous-processing-strategy | The behavior related to flow components execution is the same as flows in 4.x but it doesn’t always execute in the same thread as in 3.x. |
| transactional scope | Replaced with “try” scope. |
| username-password-filter | Removed. |
| validator | Removed. |
| weblogic-transaction-manager | Removed. |
| websphere-transaction-manager |  |
| all-strategy | Removed. |
| entry-point-resolver | Use the Java module. |
| filter | Filters are replaced by the Validations module. |
| interceptor | Interceptors are replaced with custom policies. |
| message-info-mapping | Removed. |
| point-resolver-set | Use the Java module. |
| router | Removed. |
| threading-profile | Removed. The Runtime now tunes itself. |
| transformer | Removed. Most explicit transformations are no longer needed. Use DataWeave for the corner cases. |

**===================================================================**

**Sample Migration Example:**

**===================================================================**

Mule 4 supports property placeholders either as .yaml or .properties configuration files.

1. Create a folder with the name config under /src/main/resources project directory.
2. Create a configuration file with the name configuration.yaml inside the newly created config folder.
3. Migrate property placeholders from .properties to .yaml format.

configuration.properties

http.host=0.0.0.0

http.port=8081

mysql.password=pa$$w0rd

mysql.port=3306

mysql.user=admin

mysql.database=products

mysql.host=corp.services.com

autodiscovery.api.version=1.0.0:1762946

autodiscovery.api.name=groupId:com.mulesoft.retailer.manufacturingit.apis:assetId:product-api-database

anypoint.platform.client\_id=1f702j71hu9z2x88v9vd19v7h248s589

anypoint.platform.client\_secret=87v8V47668701Dd574B0531255d6287d

configuration.yaml

http:

host: "0.0.0.0"

port: "8081"

mysql:

password: "pa$$w0rd"

port: "3306"

user: "admin"

database: "products"

host: "corp.services.com"

autodiscovery:

api:

name: "groupId:com.mulesoft.retailer.manufacturingit.apis:assetId:product-api-database"

version: "1.0.0:1762946"

anypoint:

platform:

client\_id: "1f702j71hu9z2x88v9vd19v7h248s589"

client\_secret: "87v8V47668701Dd574B0531255d6287d"

4. Replace the standard Spring element <context:property-placeholder> with the new Global Element configuration-properties.

Configuration XML for Property Placeholders in Studio 7.

|  |  |
| --- | --- |
|  | <configuration-properties file="config/configuration.yaml" doc:name="Configuration properties" / |

**5. Migrating Global HTTP Listener Configuration**

Configuration XML for Global HTTP Listener Configuration in Studio 6.

|  |  |
| --- | --- |
|  | <http:listener-config name="HTTP\_Listener\_Configuration" host="${http.host}" port="${http.port}" doc:name="HTTP Listener Configuration"/> |

The minimal configuration requires specifying host and port in the inner http:listener-connection element. We use the placeholders defined in the previous step.

Configuration XML for Global HTTP Listener Configuration in Studio 7.

|  |  |
| --- | --- |
|  | <http:listener-config name="httpListenerConfig">  <http:listener-connection host="${http.host}" port="${http.port}" />  </http:listener-config> |

## 6. Migrating API Autodiscovery Configuration

The api-platform-gw global element is required for registering an API in Anypoint Platform.

Configuration XML for API Autodiscovery Configuration in Studio 6.

|  |  |
| --- | --- |
|  | <api-platform-gw:api apiName="${autodiscovery.api.name}" version="${autodiscovery.api.version}" flowRef="api-main" create="true" doc:name="API Autodiscovery"/> |

In Mule Runtime 4.x, the apiName, version, and create attributes were removed. Just the apiId and flowRef attributes are required. apiId is generated by API Manager and visible on the API instance dashboard.

For API Autodiscovery Configuration in Mule Runtime 4.x:

1. Add the following Namespace, Schema global.xml Configuration file.

xmlns:api-gateway="http://www.mulesoft.org/schema/mule/api-gateway"

http://www.mulesoft.org/schema/mule/api-gateway http://www.mulesoft.org/schema/mule/api-gateway/current/mule-api-gateway.xsd

1. Add the required Autodiscovery Dependency Information to project pom.xml file.

<dependency>

<groupId>com.mulesoft.anypoint</groupId>

<artifactId>mule-module-autodiscovery</artifactId>

<version>4.0.0</version>

</dependency>

Configuration XML for API Autodiscovery Configuration in Studio 7.

## Migrating Global Validation Configuration

Configuration XML for Global Validation Configuration in Studio 6.

|  |  |
| --- | --- |
|  | <validation:config name="Validation\_Configuration" doc:name="Validation Configuration"/> |

Opposite to Mule Runtime 3.x, adding the Validation Module to the Mule Palette is required to proceed with the configuration.

1. In the Mule Palette, click Add Module and select Validation Module among the available modules.
2. Add a Validation Config Global Configuration Element.

Configuration XML for Global Validation Configuration in Studio 7.

|  |  |
| --- | --- |
|  | <validation:config name="Validation\_Config" doc:name="Validation Config" /> |

|  |  |
| --- | --- |
|  | <api-gateway:autodiscovery apiId="${autodiscovery.api.id}" flowRef="api-product-main" doc:name="API Autodiscovery"/> |

1. Create a **Folder src/main/resources/variables** and write DW scripts to create Flow Variables.
2. Add a Transform component to replace the logic inside message-properties-transformer to set the value of the variables.

Like:

<flow name="get-products-flow">

<ee:transform doc:name="Get Query Params" doc:id="ab756164-e1df-4fc5-8fbe-8f4f8cafc2f6">

<ee:message />

<ee:variables>

<ee:set-variable variableName="queryOffset" resource="variables/set-queryOffset-variable.dwl" />

<ee:set-variable variableName="queryLimit" resource="variables/set-queryLimit-variable.dwl" />

<ee:set-variable variableName="queryName" resource="variables/set-queryName-variable.dwl" />

<ee:set-variable variableName="queryCategory" resource="variables/set-queryCategory-variable.dwl" />

</ee:variables>

</ee:transform>

</flow>

1. Replace the Mule 3.x transactional scope with the new try scope and set the transactionalAction attribute to ALWAYS\_BEGIN
2. Create a set-httpStatus-with-XXX.dwl file under src/main/resources/variables as follows.

%dw 2.0

output application/java

---

204

1. Set httpStatus variable with the value XXX using a Transform Component for defining a NO CONTENT response code.

|  |  |
| --- | --- |
|  | <ee:transform doc:name="Set 204 HTTP Status code">  <ee:message />  <ee:variables>  <ee:set-variable variableName="httpStatus" resource="variables/set-httpStatus-with-204.dwl" />  </ee:variables>  </ee:transform> |

To return a specific HTTP Status code, instead of setting a http.status property, APIkit in Mule 4 requires setting a variable with the name httpStatus.

## Migrating Backend Flows

For each Backend Flow generated by APIkit, add a flow-ref to its implementation.

**Configuration XML for Backend Flows in Studio 6.**

<flow name="get:/product:api-config">

<flow-ref name="get-products-flow" doc:name="get-products-flow" />

</flow>

<flow name="get:/product/{id}:api-config">

<flow-ref name="get-product-by-id-flow" doc:name="get-product-by-id-flow" />

</flow>

<flow name="post:/product:application/json:api-config">

<flow-ref name="post-product-flow" doc:name="post-product-flow" />

</flow>

<flow name="delete:/product/{id}:api-config">

<flow-ref name="delete-product-flow" doc:name="delete-product-flow" />

</flow>

<flow name="put:/product/{id}:application/json:api-config">

<flow-ref name="put-product-flow" doc:name="put-product-flow" />

</flow>

**Configuration XML for Backend Flows in Studio 7.**

<flow name="get:\product:api-product-config">

<flow-ref name="get-products-flow" doc:name="get-products-flow" />

</flow>

<flow name="get:\product\(id):api-product-config">

<flow-ref doc:name="get-product-by-id-flow" name="get-product-by-id-flow"/>

</flow>

<flow name="post:\product:application\json:api-product-config">

<flow-ref name="post-product-flow" doc:name="post-product-flow" />

</flow>

<flow name="delete:\product\(id):application\json:api-product-config">

<flow-ref doc:name="delete-product-flow" doc:id="38894873-9a01-4e59-8362-5eefce5ea043" name="delete-product-flow"/>

</flow>

<flow name="put:\product\(id):application\json:api-product-config">

<flow-ref doc:name="put-product-flow" doc:id="e64cf378-26a2-4435-8d0d-269c50282b3c" name="put-product-flow"/>

</flow>