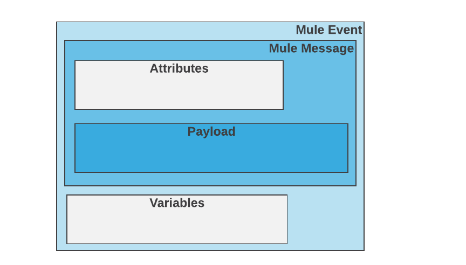
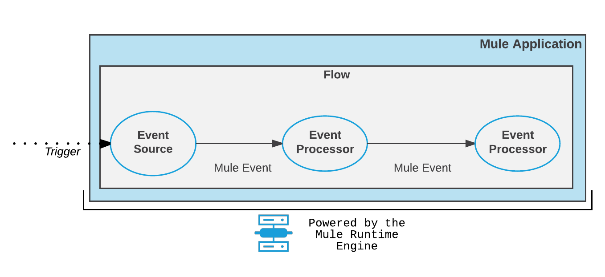
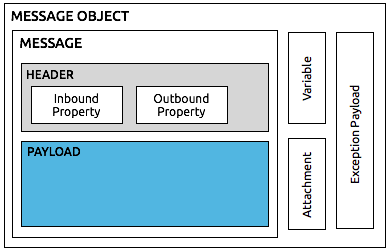
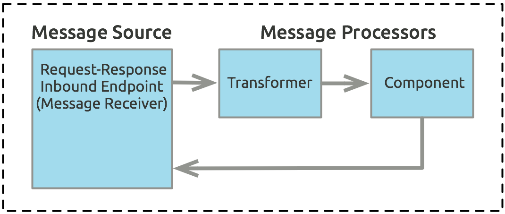
**MuleSoft Integration Professional Exam Preparation**

**PASSING SCORE IS 80%!!!!!!!!!!!!!!!!!!!!!!!!!! BE READY!**

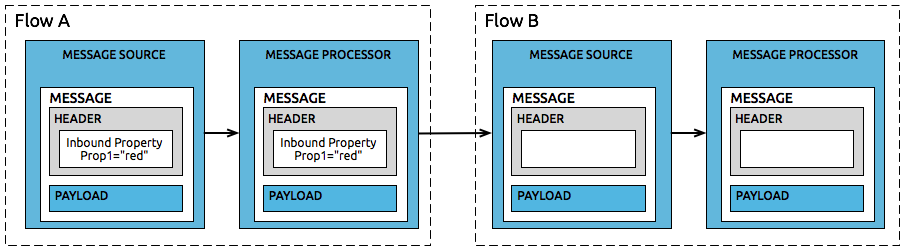
**Mule Runtime**

* Mule runtime engine (Mule)
  + Lightweight integration engine that runs mule apps
* Mule App
  + Used to connect systems,services,apis and devices using **MuleSoft’s API-led connectivity**
  + Provide functionality for message routing, data mapping, orchestration, reliability, security, and scalability
* Mule
  + Connect data to apps in Enterprise Service Bus (ESB) patterns
    - ESB (Enterprise Service Bus)
      * An Architecture
      * ESB Architecture is that you integrate different applications by putting a communication bus between them and then enable each application to talk to the bus
      * To move away from point to point
      * Adapter
        + There is an adapter between the application and the bus that marshals data between the two parties
        + Adapter is responsible for talking to the backend application and transforming data from the application format to the bus format (The data that travels on the bus is a canonical/standard format and is almost always XML)
      * Generally stateless, the state is embedded in the messages passing through the bus
      * Canonical message format
        + There is one consistent message format traveling on the bus and that every application on the bus can communicate with each other
* Throughput
  + The amount of work done
* Response Time
  + Reduce response time
* Design Center/ Studio IDE
  + Mule apps develop in design center can run in the cloud or in studio and can interface with services and databases to perform tasks, such as data conversation
  + RAML Editor
  + Support Open API Specification (OAS) 2.0
    - The importation of an OAS specification converts it to RAML
    - Share specifications using exchange
    - Import/Export specification
    - Simulate API calls
    - Test methods
    - Design center supports the reuse of api fragments and organizes dependencies.
* Runtime Manager
  + Use to deploy Mule Aps to runtimes within CloudHub and other supported platform as a service (Paas)
  + Provide dashboard for deployed app and monitored server
* CloudHub
  + Cloud-based integration platform as a service (IPaaS) for anypoint platform that enables you to run your mule apps without requiring you to provide mule runtimes or the infrastructure on which your apps run.
  + Use runtime manager to deploy mule apps to cloudhub
* Event-Driven Architecture
  + Mule works by responding to events that are initiated by external or internal resources
  + Mule processes events as messages through event processors
  + Can handle high volume of messages in batch through batch processors
* Mule Event
  + Contains the core information processed by the runtime
  + Immutable, so every change to an instance of a mule event results in the creation of a new instance
  + Represents any data event occurring in the mule environment. All data sent or received within the mule environment will be passed between components as an MuleEvent
  + Holds a MuleMessage payload and provides helper methods for obtaining the data in a format that the receiving mule component understand
  + Business event information
  +  
  + Reference <https://docs.mulesoft.com/mule-runtime/latest/about-mule-event>
* Mule Message
  + Is the data that passes through an application via one or more flows. It consists of two main parts
    - The message header, which contains metadata about the message
    - The message payload, which contains your business-specific data
  + Mule message is, itself, embedded within a mule message object
  + 
* Flow
  + Event processing components are arranged into one or more container-like components
  + Flows are sequences of Event processors.
  + 
* Event Source (Trigger)
  + First component in a flow
  + Receives a triggering event
  + Triggers (JMS, HTTP, FTP, JDBC or File, Scheduler, VM),
  + Translates these communication protocols and methods into a standard message format, which passes through the flow’s Event processors.
* Event Processors
  + Once a flow is triggered through an event source, subsequent components process the mule event as it travels through the flow
  + By default, each event processor that receives a Mule Event returns a new mule message
* DataWeave Latest Version
  + Primary language used for formulating expressions in Mule
  + You can use dataweave to access, manipulate, transform, and extract data from the mule event
* Batch Processing
  + Split messages into individual records, act on each record, then report the results and, if necessary, push the processed output to other systems or queues.
  + Best for streaming input or near real-time data integration between saas applications
* API Gateway
  + Mule Runtime includes an embedded API gateway, Enable you to apply security policies to an API, enrich incoming or outgoing messages, and add capabilities to an api without having to write any code
* Non-Blocking Execution Engine
  + The Mule execution engine is based on a non-blocking runtime
  + Task-oriented execution model that allows you to take advantage of non-blocking IO calls and to make tuning configurations in a simple way

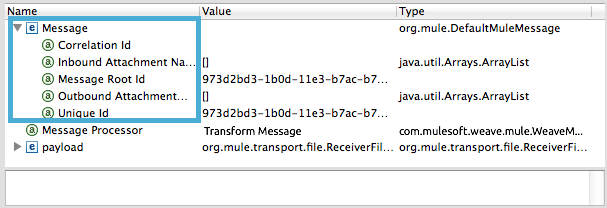
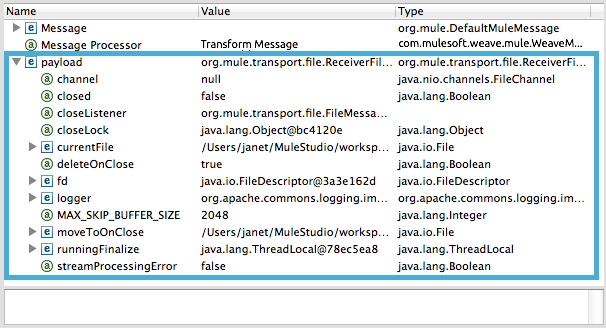
**Key Concepts**

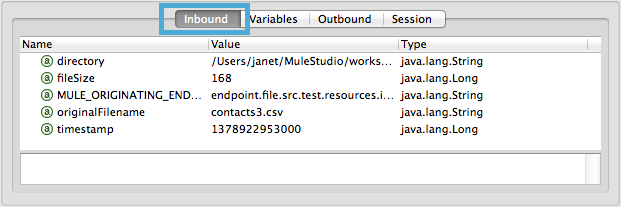
* Messages and Events
  + Event Driven Architecture (EDA)
  + Mule applications accepts and process events as messages through several message processors
  + Message processors are arranged into a Flow
  + Large or streaming messages can instead be processed as broken down into records in a batch job rather than a flow
  + Flow can call another flow as a direct reference, or may call a flow through a common communication protocol or method such as JMS, HTTP, FTP, or File
  + The called flows might be part of the same Mule application, or might be running in a separate application deployment running on another mule runtime across the network
* Flows
  + Flow is the construct within which you link together several individual processors to handle the receipt, processing, and eventual routing of a message
  + Flows are sequences of message processors
  + Request-Response
    - Mule receives the message through a request-response inbound endpoint, transforms the content into a new format, and processes the business logic in a component before returning a response via the message source.
    - 
  + Each processor corresponds to an icon in the anypoint studio GUI, which in turn represents a message source, processor, or component, which is written into the application’s XML file as an XML element
* Message Sources
  + Mule processes messages, also known as events
  + Anypoint connectors, elements which provide connectivity to a specific external source, either via a standard protocol (such as HTTP, FTP, SMTP) or a third-party API
* Message Processors
  + Mule Transformers
    - Key to exchanging data between nodes, as they allow mule to convert message payload data to a format that another application can understand
    - Content Enrichment
  + Components
    - Can be pojos, spring beans, java beans, groovy scripts,web services.
    - Can be developed in other languages
  + Filters,scopes, and routers
* Mule Expression Language (MEL) v3.9
  + Primary language for formulating expressions in Mule, allowing you to access, manipulate, and use information from the message and its environment.
* Batch Processing
  + Block of code that splits message into individual records, performs actions upon each record, then reports on the results and potentially pushes the processed output to other systems or queues.
  + Best for streaming, near real-time
  + Mule processes asynchronously

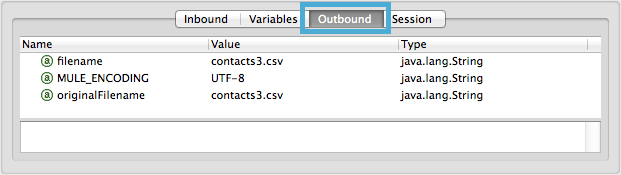
**Mule Message Structure**

* **Link:** <https://docs.mulesoft.com/mule-runtime/3.9/mule-message-structure>
* Mule message
  + Message Header, which contains metadata about the message
  + Message payload, which contains your business-specific data
  + Embedded within a mule message object
* Properties and Variables
  + The metadata contained in the message header consists of **properties** which provide useful information about the message. Contained within the message object, **variables** represent data about a message.
  + Has a **name** and a **value**
  + Property have two main scope
    - **Inbound**
      * Immutable, are automatically generated by the message source and cannot be set or manipulated by the user
      * The contain metadata specific to the message source that prevents scrambling of data formats or other processing mishaps later in the message’s lifecycle
      * A message retains its inbound properties only for the duration of the flow; when a message passes out of a flow, its inbound properties do not follow it
      * 
    - **Outbound**
      * Mutable
      * They are set during the course of a flow and can become inbound properties when the message passes from the outbound endpoint of one flow to the inbound endpoint of a different flow via a transport
      * Outbound property is applied after the message enters the flow
  + Variables (With example: variables)
    - User-defined metadata about a message. Variables have three scopes:
      * **Flow variables** apply only to the flow in which they exist
        + is global to all flows, except for flows with the transport barriers
        + You can access the value using #[flowVars.varName] or #[varName] or #[flowVars['varName']]
      * **Session variables** apply across all flows within the same application
        + is also global to all flows, it can even go past transport barriers
        + note that session variables persist across transport barriers when used with transports which support sessions
        + You can access the value using #[sessionVars.svarName] or #[svarName] or #[flowVars['svarName']]
      * **Record variables** apply to only to records processed as part of a batch
      * **Property** 
        + once a message hits an outbound endpoint, all properties in the outbound scope are sent with the message, in the form of transport specific metadata (e.g http header)
        + to access the property that was recently added within the flow, use #[message.outboundProperties]
        + to access the property that passed to transport barrier, use #[message.inboundProperties]
    - Temporary pieces of information about a message that are meant to be used by the application that is processing it, rather than passed along with the message to its destination
    - Reference: <https://blogs.perficient.com/2017/02/02/mule-variable-scopes-and-passing-global-values-with-mule-registry/>
    - Property,variable,session variable
      * set, copy or remove outbound properties
    - Record variable
      * set or remove variables on a record in a batch
  + Message Payload
    - the most important part of the mule message because it contains the data your mule application processes
  + Message Enricher
    - to enrich the message/transform
  + DataSense explorer
    - This is useful to know the names of variables and properties that are available at that point, as well as the payload’s internal structure

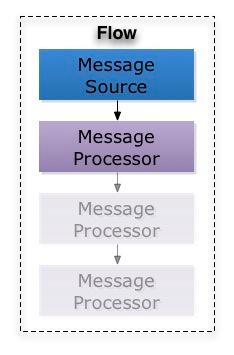
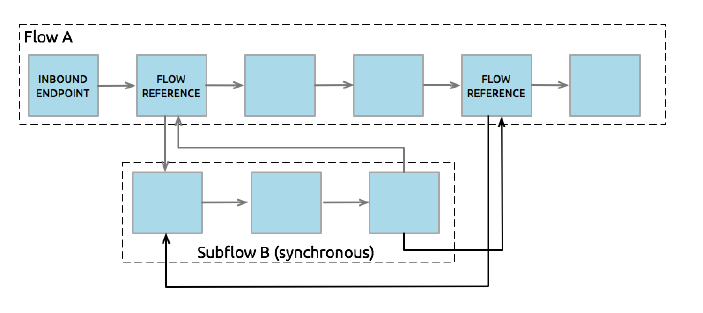
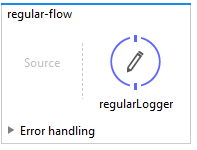
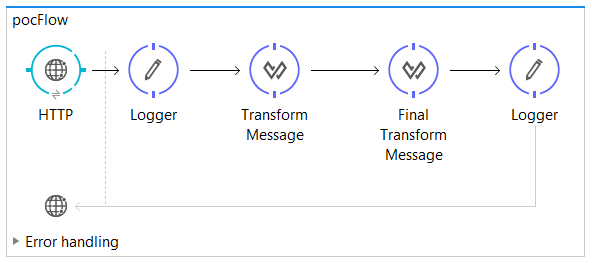
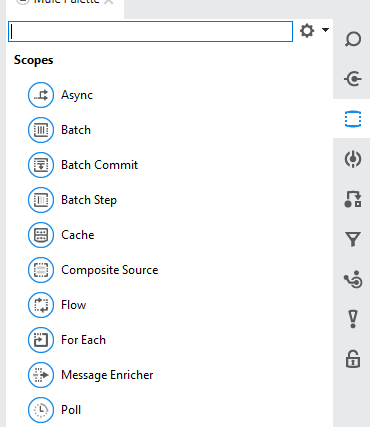
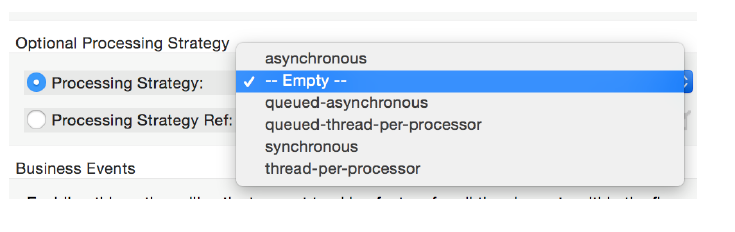
**Mule Message Transformation (With example muleMessageTransformation)**

* **Link:** <https://docs.mulesoft.com/mule-runtime/3.9/message-state>
* To better understand how Mule message processors act upon messages, it is useful to examine a message before and after it is processed. E.g JSON to Java Object, Map to CSV etc.
* **Message**
  + Contains data from the header of the message (i.e metadata)
* **Message Processor**
  + Indicates a Value of transform message; the Message Processor item indicates the next message processor in the flow that the message will encounter
* 
* **Message Payload**
  + Contains the payload of the message, or rather, the data that the mule application processes.
  + 
* **Properties**
  + Inbound
    - include metadata about the payload
    - read-only and cannot be added, removed or copied by message processors in the application

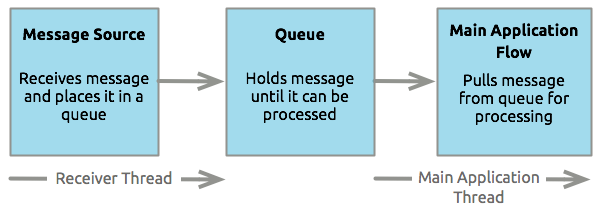
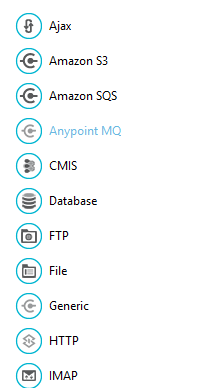
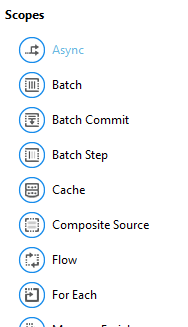
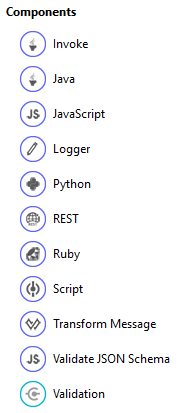
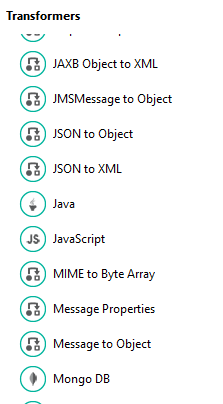
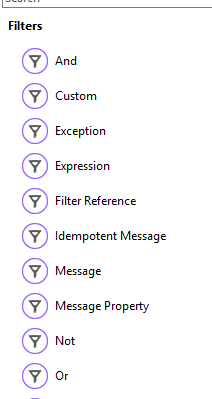
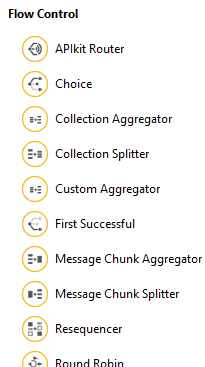
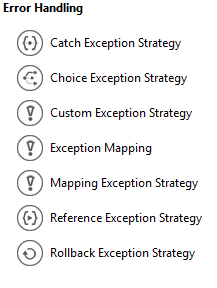
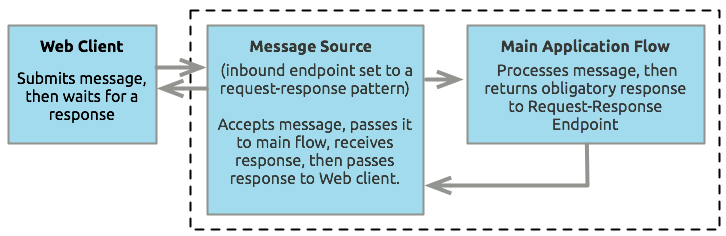
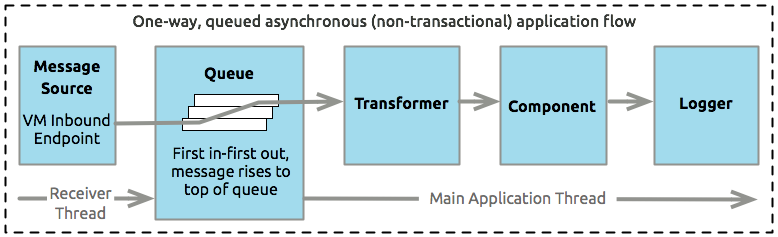
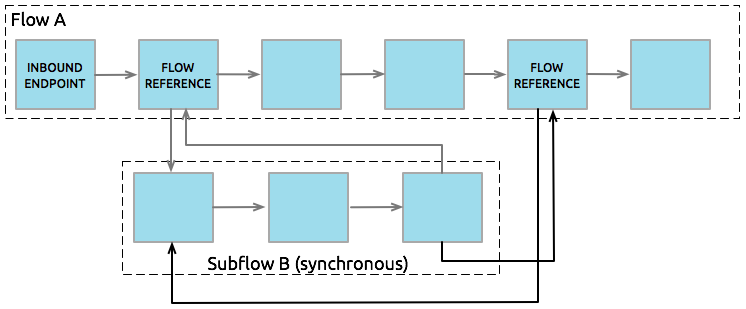


* + Outbound
    - Indicate similar information about the payload, and can be removed or copied by message processors in the application.
    - 
* **Variables**
  + Visual debugger displays any variables or session variables included in the message object as it encounters the transform message components

**Flows**

* Simple yet very flexible mechanism that enables orchestration of services using the sophisticated message flow capabilities of mule.
* most versatile and powerful integration mechanism
* Flows are valuable in many situations, including;
  + Simple integration tasks
  + Scheduled data processing
  + Connecting cloud and on-premise applications
  + Event processing where multiple services need to be composed
* Essence just a chain of message processors
* Has a message source
* Mule automatically sets a flow to be Synchronous or queued-asynchronous
* A flow is set to asynchronous if the message source is request-response
* Otherwise, a flow is set to queued-asynchronous. The message source is not expecting a response
* 
* <flow name="">
* - 0..1 MessageSource
* - 1..n MessageProcessor(s)
* - 0..1 ExceptionStrategy
* </flow>
* **References:** <https://www.javarticles.com/2017/03/mule-message-exchange-pattern-example.html>
* <https://dzone.com/articles/mule-sub-flows-processing-strategy-and-one-way-end>
* **One-way**
  + won’t return any response
  + **Inbound endpoints**
    - Wont return a response to the caller
  + **Outbound endpoints**
    - Wont wait for a response from the callee
* **Request-response**
  + will return a response in the same thread
  + **Inbound endpoints**
    - Return response
  + **Outbound endpoints**
    - Will wait
* **Inbound and Outbound endpoints**
  + Mule is typically divided into two parts (Source and Process)
  + **Source = Inbound endpoints**
  + **Process = if endpoint is placed here then we call it Outbound**
* **SubFlows**
  + Executed exactly as if the processors were still in the calling flow
  + Always run sync in the same thread
  + Inherit the processing and exception strategies of the flow that triggered its execution
  + 
* **Flows**
  + They can have their own processing and exception strategies
  + The can be sync or async
  + flows without message sources are sometimes called **private flows**
    - 
  + **Private Flows**
    - Cannot be accessed from outside the JVM via a **mule endpoint** because it has no message source defined
    - Only used if they are referenced from another construct running in the same mule instance
* **Staged Event Driven Architecture (SEDA)**
  + The architecture upon which mule was built
  + Decouples receiving, processing, and dispatching phases
  + Supports higher levels of parallelism in specific stages of processing
  + Allows for more-specific tuning of areas within a flow’s architecture
* **Synchronous flows**
  + When a flow receives a message, all processing, including the processing of the response, is done in the same thread
  + Uses only the message source’s thread pool
  + The flow’s thread pool is elastic and will have one idle thread that is never used
  + Tunning for higher-throughput happens on the connector receiver’s level
  + 
* **Queued-asynchronous flows**
  + Decouples and uses all 3 thread pools
  + Uses queues, whose threads drop messages off for the subsequent pool’s thread to pick up
  + Pools, queues, and behaviors of this strategy are configurable
  + By default, the flow thread pool has **16 threads**.
* **Scope Component**
  + 
* **Setting the flow processing strategy**
  + The flow processing strategy is automatically set (--Empty--)
  + It can be changed in the flow’s properties view
  + This is usually done when a custom queued-asynchronous profile has been created for tuning application performance
  + 
* **Using VM endpoints in flows – The java Virtual Machine (VM) transport**
  + **Ref:** <https://dzone.com/articles/mule-private-flow-versus-vm-transport>
  + Intra-JVM communication between Mule flows
  + Each app in a mule instance has its own, unique set of VM endpoints
  + can only handle communications within an app or between apps in the same domain
  + by default it uses in-memory queues but can optionally be configured to use persistent queues
  + Achieve higher levels of parallelism in specific stages of processing
  + Allow for more-specific tuning of areas within a flow’s architecture
  + Call flows in other applications that are in the same domain
  + VM endpoints enable message redelivery strategies to be configured in your exception handling blocks, which is not possible with flow-refs. VMs are able to do this because they internally use a queue to hold messages while flow-refs are similar to simple method calls.
  + Use a flow-ref by default, but don’t hesitate to use VM transports if you need redelivery of messages

**Flow Architecture in a Mule Application (followed next step link start here)**

* **Key Points**
  + At the simplest level, Mule applications accept one message at a time, processing received messages in the order in which they are received.
    - Sometimes, the mule application can return an **altered or replacement message** to the source of the original message
    - Additionally or instead, the application **can send the message** in its original or altered form to one or more third parties
    - in still other cases, Mule can **decline to process the message** if it has not met specific criteria
* **Message Sources**
  + Which receives messages from one or more external sources, thus triggering a flow instance. Each time it receives another message, the message source triggers another flow instance
  + Sometimes the message source immediately places the incoming message into a queue. This allows the message source to close the receiver thread it used to accept the message, and immediately open another thread to accept another incoming message
  + 
* **Message Processors**
  + **Connectors**
    - The provide a means for Mule applications to communicate with the outside world. Connectors often serve as message sources. But they can also appear elsewhere in a flow
    - 
  + **Scopes**
    - Scopes are Mule components that can wrap a group of operations within a flow to define a fine grained behaviour for them to apply
    - Scopes do nothing on their own, they must have child components to act on
    - 
  + **Components**
    - They allow you to enhance a flow by attaching functionality such as logging, or displaying output. Groovy, Java, Invoke, Transform Message, Validator. More use on validations
    - 
    - Custom business logic
  + **Transformers**
    - The enhance or alter the message payload, properties, variables, or attachments
    - 
  + **Filters**
    - Singly and in combination, they determine whether a message can proceed through an application flow.
    - 
  + **Flow Controls**
    - The specify how messages get routed among the various message processors within a flow. They can also process messages (that is, aggregate, split, or resequence) before routing them to other message processors.
    - 
  + **Error Handlers**
    - They specify various procedures for handling exceptions under various circumstances
    - 
  + **Miscellaneous**
    - This special category currently contains just one member: **Custom Business Event Processor**
* **Exchange Patterns**
  + Request-Response
    - When an inbound endpoint-based connector such as HTTP or VM are configured for a request-response exchange pattern, it effectively becomes a hybrid inbound-outbound endpoint
    - 
  + One-way
    - not expected to provide any response to the original sender
* **Processing Strategies**
  + Determines how mule executes the sequence of message processors in your application
  + e.g when the message source is configured with a **request-response exchange pattern**, mule sets the processing strategy to **synchronous,** which means that the entire flow gets executed on a single processing thread, thus ensuring that the entire sequence of message processors executes, and the client receives a response, as expected
  + when the flow is configured for a **one-way exchange pattern** and is non-transactional(that is no response to the original message sender is required, and it isn’t necessary to verify that all steps in the flow have been completed), mule sets the processing strategy to **queued asynchronous,** which has the potential to raise flow throughput
  + 
* **Exception Strategies**
  + An **exception strategy** determines how mule responds if and when an error occurs during the course of message processing. In the simplest case, the error is simply logged to a file.
* **Flows and Subflows**
  + Ideally suited for code reuse, so you can write a particular block of code once
  + Subflows inherit the processing strategies and exception strategies of the flow that triggers it, which means you don’t have to define these same configuration details again when building a subflow
  + 

**Mule Components**

* Components are message processors which execute business logic on messages. They enable you to perform specific actions without writing any mule-specific code. You can drop a component -a POJO, Spring bean, java bean or script. Into any flow to perform almost any customized task within your mule application.
* **HTTP Static Resource Handler (With example httpStaticResourceHandler)**
  + To easily server up static content when called (.html, pdf etc)

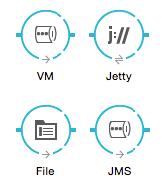
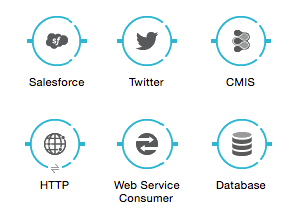
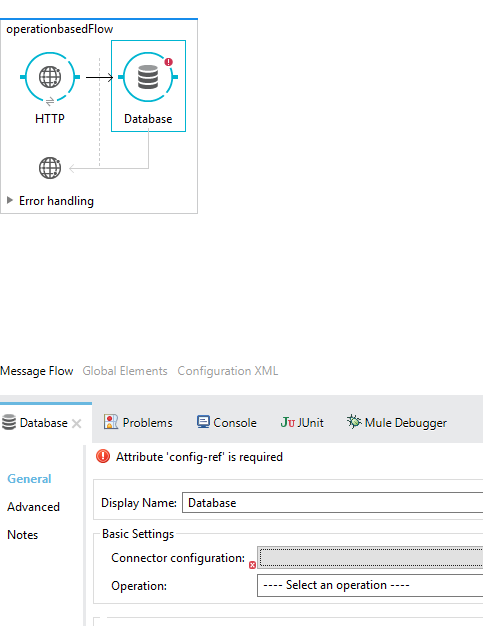
**Transformers**

* In a Mule flow, a transformer prepares a message for further processing by enhancing or altering the contents of the message properties, variables, or payload. Data transformation is one of the most powerful functionalities of Mule. Json To Object, etc.
* **DataWeave Transformer (Transform message)**
  + Converts and maps data
  + Graphically map fields by dragging one attribute to another, just like you were able to with the now deprecated DataMapper
  + Support DataSense
  + Allows you to retrieve the message payload in a particular format, e.g #[message.payloadAs(java.lang.String)]

**Mule Filters, Scopes, and Routers**

* **Filters**
  + Evaluate a message to determine whether it can proceed through a flow.
  + Simplest filters (and, or, and not)
* **Scopes**
  + Work to encapsulate other message processors so that they function as a single unit.
* **Routers**
  + Or **flow controls**, to direct or otherwise control messages within a flow
  + Splitters, resequencers, or aggregators, splitting messages into individual items for processing, resequencing message content, or aggregating split messages.

**Anypoint Connectors**

* Receive or send messages between Mule and one or more external sources, such as files, databases, or Web services.
* Connectors can act as message sources by working as inbound endpoints, they can act as a message processor that performs an operation in the middle of a flow, or they can be at the end of the flow and act as a the recipient of the final payload data.
* Connectors in Mule are either **endpoint-based** or **operation-based.**
* **Endpoint-based**
  + Follow either a one-way or request-response exchange pattern and are often (nut not always) named and based around a standard data communication protocol, such as FTP and SMTP
  + Configured as either inbound or outbound endpoints in a flow
  + Serve as a message source or mid-flow or at the end of flows, and send information to external systems
  + 
* **Operation-based** 
  + follow an **information exchange pattern** based on the operation that you select and are often (but not always) named and based around one or more specific third-party APIs.
  + 
  + e.g sfdc:query or sfdc:upsert-bulk
  + 
* **Global Connector Configuration**
  + Some connectors require that connection information such as username, password, and security tokens be configured in a global element rather than at the level of the message processor within the flow. Many connectors of the same type in an application can reference the connector configuration at the global level.
  + For operation-based connectors, the global connector configuration is mandatory, but for most endpoint-based connectors it is optional
  + **Global element** 
    - **<connectorName>:config**
    - **E.g HTTP Connector, Salesforce**
* <http:listener-config name="HTTP\_Listener\_Configuration" host="localhost" port="8081" doc:name="HTTP Listener Configuration"/>
* <http:request-config name="HTTP\_Request\_Configuration" host="localhost" port="8082" doc:name="HTTP Request Configuration"/>
  + **Operation-based connector**
    - **<connectorName>:connector**
    - **E.g JMS Connector**
* <jms:activemq-connector name="Active\_MQ" username="test" password="test" brokerURL="tcp://localhost:61616" validateConnections="true" doc:name="Active MQ"/>

**Global Elements**

* is a reusable object containing parameters that any number of elements in a flow can share.
* use the global element to apply configuration details to multiple **local** elements in flows.
* Ensure consistency across flow elements
* **Name Attribute for security Manager Global Element**
  + **Name** attribute for the security manager global element is now required in Anypoint studio as of version 6.0.0
* **Append a String**
  + If you find that you need to append the same string multiple times at various points in a flow, you migt store such a piece of functionality inside a global element for a **Transformer** of type **Append String**
  + If you are coding by hand, notice the global element is defined outside and above the <flow> that references it
  + **Todo** <https://docs.mulesoft.com/mule-runtime/3.9/mule-high-availability-ha-clusters>

**General**

• Explain basic MuleSoft implementation and design concepts

• Track data movement through an application

**Basics**

• Identify when to use and use flow variables and session variables

• Write Mule expressions

• Define Mule properties and create properties files

**HTTP Connector**

• Create and configure inbound and outbound HTTP endpoints

• Use HTTP and HTTPS

• Define HTTP content-type and explain its effect on browser types

**Flows**

• Use flows, sub-flows, and flow references

• Explain the differences between inbound and outbound endpoints

• Configure flow processing strategies

• Code and test exchange patterns (including request-response and one-way)

• Test Mule applications using JUnit and MUnit cases

• Send a Mule message from a test class to a Mule application

**Flow Control**

• Use splitters, aggregators, and multicast routers

• Use the For-each scope

• Use filters

**Error Handling**

• Debug flows and expression handlers

• List the different exception strategies that are available

• Use exception strategies and explain how they affect flows and sub-flows

• Change and return a message from an exception strategy

• Configure global application exception handling

• Use routers (including First Successful and Until Successful) to handle potential error conditions

**Transformations with DataWeave**

• Write DataWeave scripts to convert JSON, XML, and Java data structures to different data structures and data types

• Use DataWeave operators

• Define and use custom data types

• Apply correct DataWeave syntax to coerce data types

• Apply correct DataWeave syntax to format strings, numbers, and dates

• Call Mule flows from a DataWeave script

• Call global MEL functions from a DataWeave script

**Web Services**

• Implement REST services with GET, POST, PUT, and DELETE methods

• Use annotations on REST methods to create unique signatures

• Create REST clients and working with dynamic endpoints

• Publish and consume SOAP messages

• Use CXF interfaces to create service definitions

• Extend interfaces to create CXF implementations

**Scopes**

• Configure and use batch processing

• Use the Cache Scope to store and reuse frequently called data

• Create and manage caching strategies

• Use Enrichers to enhance a Mule message

**Deployment**

• Explain the general concepts and benefits for building Mule clusters

• Manage runtime clusters

• Use queues to distribute application flows for processing in clusters

• Describe how clustering supports various Mule transport mechanisms

• Deploy applications to customer-hosted Mule runtimes

• Deploy applications to CloudHub

• Organize Spring properties and Spring property file configuration

**Java Components**

• Create and test Java custom components and integrating them into flows

• Use advanced Java concepts to invoke service calls for passing Mule messages

• Create custom filters with Java

• Configure Java components to be prototypes or singletons

• Use the default entry point resolver with Java components

**Connectors and Transports**

• Configure and use Database connectors

• Explain how Database inbound and outbound endpoints differ and their limitations

• Configure JMS connectors for two-way communications, temporary queues, and object serialization over transports

• Use back channels and creating two-way communication through JMS connections

• Describe how JMS uses correlation IDs

• Use VM Transport to control how messages are sent and received by components in a system

• Use VM Transport for communication between Mule flows

• Explain queue usage with VM Transport and configuration structure

• Configure and use File and FTP connectors

**Transactions**

• Explain transaction management

• Identify which endpoints support transactions

• Manage and configure resource transactions for inbound and outbound messages

• List the various transaction types and usage techniques

**References:**

* MuleSoft Certified Developer – Integration Professional (Mule 3) Exam Curriculum
  + <https://training.mulesoft.com/exam/mcd-pro>
* MuleSoft Integration Professional insight
  + <https://profit-online.pl/2018/07/mulesoft-integration-professional-insight/>
* Mule Runtime v3.9 (Make sure to select the correct version v3.9)
  + <https://docs.mulesoft.com/mule-runtime/latest/>
* Runtime Manager
  + <https://docs.mulesoft.com/runtime-manager/>
* Anypoint Exchange for Mulesoft project examples
  + <https://www.mulesoft.com/exchange/?type=example>
* Useful blogs
  + <https://javastreets.com/>
  + <https://www.jerney.io/author/jerney/>
  + <https://blogs.mulesoft.com/tag/community-roundup/>