PassQueen
a.k.a dumpsqueen.com

MuleSoft Certified Developer - Integration and API Associate (Mule 3)

Mulesoft MCD-ASSOC

Version Demo

Total Demo Questions: 12

Total Premium Questions: 121

Buy Premium PDF

https://passqueen.com

support@passqueen.com

QUESTION NO: 1

Which two statements correctly describe Mule 3 exception strategies?

(Choose two.)

A. A Catch Exception Strategy handles an exception and allows the flow to complete normally

B. A Rollback Exception Strategy rolls back an active transaction

C. A Catch Exception Strategy always propagates the exception to the calling flow

D. A Rollback Exception Strategy commits an active transaction

E. Exception strategies can only be configured inside a subflow

ANSWER: A B

Explanation:

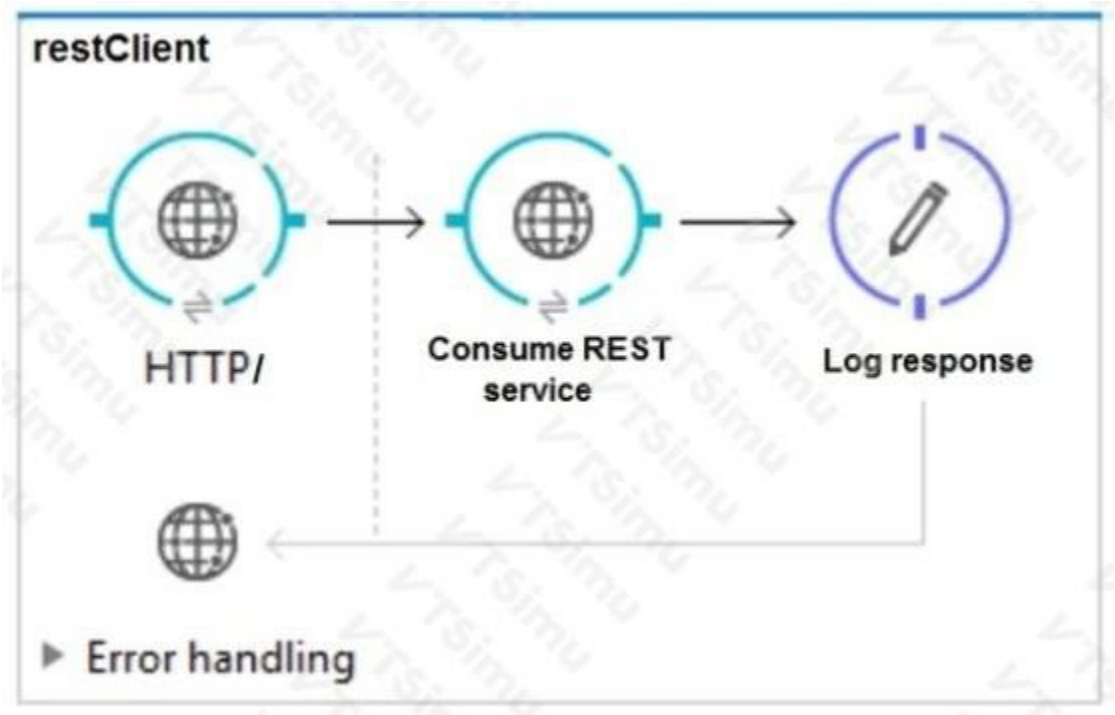
A **Catch Exception Strategy** handles an exception and allows the flow to complete normally after the strategy finishes. When a transaction is active, handling the exception with a Catch Exception Strategy causes the transaction to commit. A **Rollback Exception Strategy** rolls back an active transaction and propagates the exception to the flow that invoked the current flow.

These strategies provide different outcomes for recoverable handling and transactional failure. A global exception strategy can be defined for reuse, while a flow can define its own local exception strategy when specialized behavior is required. See MuleSoft documentation on Mule 3 error handling.

QUESTION NO: 2

Refer to the exhibits. The http:request has failed with a Tiemout exceeded error. What

HTTP Request parameter must be modified to resolve this error?



```
********************************************************************************
Message         : Error sending HTTP request to http://localhost:8082/.
Payload         : (NullPayload)
Payload Type        : org.mule.transport.NullPayload
Element         : /restClient/processors/0 @ mcd-assoc-4.01.02-ram-rest-consumer-error:rest-
consumer-error.xml:25
Element XML         : <http:request config-ref="HTTP_Request_Configuration_localhost8082"
path="/" method= "GET" doc:name=
--------------------------------------------------------------------------------
------------

Root Exception stack trace:
java.utl.concurrent.TimeoutException: Timeout exceeded

    at com.ning.http.client.providers.grizzly.GrizzlyAsyncHttpProvider.timeout
(GrizzlyAsyncHttpProvider.java:433)
    at com.ning.http.client.providers.grizzly.GrizzlyAsyncHttpProvider$3.onTimeout
(GrizzlyAsyncHttpProvider.java:281)
    at org.glassfish.grizzly.utils.IdleTimeoutFilter$DefaultWorker.doWork
(IdleTimeoutFilter.java:402)
    at org.glassfish.grizzly.utils.IdleTimeoutFilter$DefaultWorker.doWork
(IdleTimeoutFilter.java:381)
    at org.glassfish.grizzly.utils.DelayedExecutor$DelayedRunnable.run(DelayedExecutor.java:158)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
    at java.lang.Thread.run(Thread.java:745)

********************************************************************************
```

A. Client Certificate Timeout

B. Transaction Timeout

C. Connect Idle Timeout

D. Response Timeout

ANSWER: D

Explanation:

Response Timeout is the HTTP Request setting that controls how long Mule waits for the target server to return an HTTP response after the request has been sent. A timeout-exceeded failure in this scenario means the remote endpoint has not completed its response within the configured response-wait interval. Increasing this value to a duration appropriate for the downstream service's expected processing time allows Mule to continue waiting rather than terminating the request prematurely. The value should be selected from measured response-time expectations and the integration's end-to-end service-level requirements, rather than simply set to an unnecessarily large value. Mule's HTTP Request operation exposes a response timeout specifically to define this maximum response wait, while connection-related settings govern establishing or maintaining the network connection rather than the duration of remote request processing. See MuleSoft's HTTP Request connector configuration reference at HTTP Request Connector Reference and the Mule runtime HTTP connector documentation at HTTP Connector Documentation.

QUESTION NO: 3

What are the three phases of a Mule batch job?

(Choose three.)

A. Load and Dispatch

B. Process

C. On Complete

D. Choice

E. Rollback

ANSWER: A B C

Explanation:

A Mule 3 batch job has three phases: **Load and Dispatch**, **Process**, and **On Complete**. In the Load and Dispatch phase, Mule creates batch records from the input message and places them in the batch job queue. During the Process phase, Mule processes each record through the batch steps, potentially using concurrent processing. The On Complete phase runs after all batch records have finished processing and is commonly used for reporting or final aggregation.

A Choice phase and a Rollback phase are not batch job phases. See the Mule 3 Batch Processing documentation.

QUESTION NO: 4

What execution model is used by For Each and Batch scopes?

A. For Each is single-threaded and Batch is multi-threaded

B. Batch is single-threaded and For Each is multi-threaded

C. Both are multi-threaded

D. Both are single-threaded

ANSWER: A

Explanation:

For Each is single-threaded and Batch is multi-threaded. In Mule 3, a For Each scope splits a collection into individual messages and routes them through its contained message processors sequentially. Each iteration must complete before processing proceeds to the next item, so it is appropriate when ordering and straightforward, synchronous iteration are needed. The scope preserves the original message context and aggregates the iteration results after the collection has been processed.

Batch processing is designed for handling large collections as records and uses a batch job execution model. After input records are loaded, Mule can process multiple records concurrently during the batch processing phase. This parallelism enables higher throughput and can be configured through batch settings such as maximum concurrency. Batch jobs also provide record-level processing, queueing, and reporting capabilities that distinguish them from simple sequential collection iteration. MuleSoft's Mule 3 documentation describes the sequential behavior of the For Each scope and the concurrent record-processing behavior of batch jobs: For Each Scope documentation and Batch Processing documentation.

QUESTION NO: 5

Which three attributes can be defined for a RAML query parameter?

(Choose three.)

A. type

B. required

C. default

D. protocol

E. method

ANSWER: A B C

Explanation:

A RAML query parameter can define its **type**, whether it is **required**, and a **default** value. The type constrains the allowed data type, such as string, integer, boolean, or a custom type. The required attribute specifies whether a client must provide the parameter. A default value documents or supplies the value to use when the parameter is omitted.

Query parameters are declared beneath the `queryParameters` node for a resource method in RAML. They are part of the method contract and are distinct from the HTTP method and protocol used by the API. See the RAML 1.0 specification.

QUESTION NO: 6

A Transform Message component receives the XML payload:

```
<order orderId= 'PO1234'>
    <customer>
        <customerName>Alice Green</customerName>
        <country>UK</country>
    </customer>
    <customer>
        <customerName> Bob Brown</customerName>
        <country>AU</country>
    </customer>
</order>
```

What is the DataWeave expression to output an array of the two customerName elements?

A. payload.order.*customer[].customerName

B. payload.order.*customer.customerName

C. payload.order.customer[].customerName

D. payload.order.customer.customerName

ANSWER: B

Explanation:

`payload.order.*customer.customerName` is correct because it navigates the XML structure from the payload root to the `order` element, then uses DataWeave's multivalue selector, `*customer`, to select every child element named `customer`. In the supplied XML, this selection contains the two customer records. Applying `.customerName` to that multivalue result extracts the `customerName` child from each selected customer, producing an array containing the two names in their source-document order.

DataWeave selectors are designed to traverse XML using element names as object keys. The multivalue selector is especially important where an XML element can occur repeatedly: it returns all matching occurrences rather than addressing only a single child. Subsequent field selection is applied across that collection, which is why no explicit loop is needed to obtain the requested array. This concise selector expression is appropriate in a Transform Message component because DataWeave represents parsed XML as a navigable data structure while preserving repeated elements as a collection for this kind of transformation. See MuleSoft's DataWeave selectors documentation and its XML format documentation.

QUESTION NO: 7

Refer to the exhibits. A Mule application is configured to use the globalErrorHandler exception handler.

When the flow is executed, a request is made to a host that iscurrently offline and a Java exception is thrown with the message "Error sending HTTP request to http://offline.bad:80/ ".

What response is returned to a web client request to postToOfflineHostFlow's HTTP Listener?

A. AFTER

B. BEFORE

C. Errorsending HTTP request to http://offline.bad:80/

D. GLOBAL ERROR

ANSWER: D

Explanation:

GLOBAL ERROR is returned because the failed outbound HTTP request raises an exception that is handled by the flow's referenced `globalErrorHandler`. In Mule 3, a catch exception strategy handles an exception locally rather than allowing it to propagate back through the inbound endpoint as an unhandled error. The handler in the exhibit sets the message payload to `GLOBAL ERROR`. Since the HTTP Listener uses the resulting Mule message payload as its response body, the client receives that replacement payload after the error handler completes.

The Java exception message identifies the connection failure that triggered error handling, but it is not automatically the response body when a catch exception strategy takes control and explicitly supplies a new payload. The normal processing path is interrupted at the outbound request; the error-handling path determines the final message returned by the listener. MuleSoft's Mule 3 documentation describes how catch exception strategies process errors and allow a flow to continue with a handled message: Mule Runtime 3 Error Handling. See also the Mule 3 guide to configuring exception strategies: Catch Exception Strategy.

QUESTION NO: 8

Which two statements are true about session variables in Mule 3?

(Choose two.)

A. They are available to subsequent flows that process the same message

B. They can persist across a transport barrier

C. They are removed when the current flow ends

D. They can only be accessed through flowVars

E. They are read-only after they are created

ANSWER: A B

Explanation:

Session variables are associated with the Mule message and remain available as the message moves through multiple flows. They can also persist across transport barriers, unlike flow variables, which are scoped to the flow in which they are created and are not propagated across a transport boundary.

A session variable is accessed in MEL through the `sessionVars` map, such as `#[sessionVars.customerId]`. Session variables should be used carefully because their broader lifetime can make message state harder to manage than flow-scoped data. See MuleSoft documentation on Mule message scopes and variables and the Mule Expression Language.

QUESTION NO: 9

A Mule message payload contains the following JSON object:

```
{ "customer": { "id": "C100", "name": "Maria" } }
```

What MEL expression accesses the customer ID?

A. #[payload.id.customer]

B. #[message.inboundProperties.customer.id]

C. #[payload.customer.id]

D. #[flowVars.customer.id]

ANSWER: C

Explanation:

`#[payload.customer.id]` is correct because the payload is an object containing a `customer` object, which in turn contains the `id` field. MEL supports dot notation to access nested object properties when their names are valid identifiers.

The expression first accesses the message payload, then the nested `customer` object, and finally its `id` value. It evaluates to `C100` for the stated payload. See the MuleSoft Mule Expression Language reference.

QUESTION NO: 10

Which three types of assets can be shared through Anypoint Exchange?

(Choose three.)

A. API specifications

B. Connectors

C. Templates

D. Runtime worker instances

E. Client secrets

ANSWER: A B C

Explanation:

API specifications, **connectors**, and **templates** can be published to Anypoint Exchange for discovery and reuse. API specifications publish an API contract for consumers. Connectors package reusable connectivity to external systems, and templates provide reusable implementation patterns that can accelerate development.

Making these assets discoverable supports an application network by allowing teams to consume governed integration capabilities rather than repeatedly building equivalent assets. Runtime worker instances and API client credentials are managed through deployment and API management capabilities, not published as Exchange assets. See the Anypoint Exchange documentation.

QUESTION NO: 11

Which three HTTP methods are idempotent?

(Choose three.)

A. GET

B. POST

C. PUT

D. DELETE

E. PATCH

ANSWER: A C D

Explanation:

GET, **PUT**, and **DELETE** are idempotent HTTP methods. An idempotent request can be made one or more times with the same intended effect on the server state. Repeating a GET does not change the resource state. Repeating a PUT replaces the target resource with the same representation, and repeating a DELETE leaves the target resource deleted even if a later request returns a different status.

POST is not generally idempotent because repeating a POST to a collection can create multiple resources. PATCH is not inherently idempotent because its effect depends on the patch operation defined by the API. See RFC 9110: Idempotent Methods.

QUESTION NO: 12

An SLA based policy has been enabled in API Manager. What is the next step to configure the API proxy to enforce the new SLA policy?

A. Add new property placeholders and redeploy the API proxy

B. Restart the API proxy to clear the APIpolicy cache

C. Add new environment variables and restart the API proxy

D. Add required headers to the RAML specification and redeploy the new API proxy

ANSWER: D

Explanation:

Add required headers to the RAML specification and redeploy the new API proxy is correct. An SLA-based policy identifies the calling client application through its client credentials, normally supplied as request headers such as `client_id` and, when configured, `client_secret`. Declaring these headers in the RAML contract makes the API's required client-identification inputs explicit to consumers and ensures the generated proxy configuration is based on the updated API definition. Redeploying the API proxy then applies that updated definition so requests reaching the proxy include the information needed for API Manager to associate them with a client application and evaluate the applicable SLA tier, including its rate-limit or quota entitlement. This keeps the published API contract, client application credentials, and gateway enforcement behavior aligned. MuleSoft describes how client applications provide credentials for client-ID-based policy enforcement and how SLA tiers are applied to those applications in the API Manager documentation: Client ID-based policies and Client applications and SLA tiers.