DUMPSARENA

MuleSoft Certified Developer - Level 2 (Mule 4)

Mulesoft MCD-Level-2

Version Demo

Total Demo Questions: 8

Total Premium Questions: 89

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Designing Mule Applications	24
Topic 2, Building Mule Applications	37
Topic 3, Testing Mule Applications	9
Topic 4, Deploying Mule Applications	13
Topic 5, Troubleshooting Mule Applications	6
Total	89

QUESTION NO: 1

A flow inserts an order into a database and then invokes an external HTTP service. The business requirement is that the database insert must be rolled back if the HTTP service fails and the error is propagated.

Which design should be used?

A. Place the HTTP Request operation in an Async scope after the database insert.

B. Place both operations in a transactional Try scope and propagate the HTTP error.

C. Use an On Error Continue handler after the HTTP Request operation.

D. Store the database connection details as secure properties before the insert operation.

ANSWER: B

Explanation:

Place the database insert and HTTP Request operation inside a Try scope configured to begin a transaction, and use an On Error Propagate handler for the failure path. When the HTTP Request fails and the error is propagated, the transaction rolls back the database work performed within that transactional scope.

An On Error Continue handler treats the owner flow as successfully completed, which is unsuitable when the database insert must be rolled back because the downstream call failed. An Async scope also cannot provide the required synchronous transactional outcome.

QUESTION NO: 2

A mule application exposes and API for creating payments. An Operations team wants to ensure that the Payment API is up and running at all times in production.

Which approach should be used to test that the payment API is working in production?

A. Create a health check endpoint that listens on a separate port and uses a separate HTTP Listener configuration from the API

B. Configure the application to send health data to an external system

C. Create a health check endpoint that reuses the same port number and HTTP Listener configuration as the API itself

D. Monitor the Payment API directly sending real customer payment data

ANSWER: A

Explanation:

Create a health check endpoint that listens on a separate port and uses a separate HTTP Listener configuration from the API. A production health check should provide a lightweight, safe request that an operations monitoring system can invoke repeatedly without creating, changing, or exposing payment data. The endpoint can return a successful HTTP response only when the Mule application is running and able to serve the health-check flow. Monitoring software can then alert the Operations team when the endpoint is unavailable or returns an unhealthy response.

Using a dedicated listener configuration and port isolates operational monitoring traffic from the Payment API's public listener and its API-specific policies, routing, and workload. This makes the health probe dependable as an operational contract and avoids using a real payment request as an availability test. The health-check flow should remain deliberately small, fast, and side-effect free; where appropriate, it can also validate essential dependencies needed for the application to perform its intended service. Mule runtime provides health-check capabilities for runtime and application monitoring, while Runtime Manager provides facilities for monitoring deployed applications and their operational state. See the Mule Health Check API documentation and Runtime Manager monitoring documentation.

QUESTION NO: 3

Mule application A is deployed to CloudHub and is using Object Store v2. Mute application B is also deployed to CloudHub.

Which approach can Mule application B use to remove values from Mule application Aâ€™S Object Store?

A. Object Store v2 REST API

B. CloudHub Connector

C. Object Store Connector

D. CloudHub REST API

ANSWER: A

Explanation:

Object Store v2 REST API is correct because it provides a management interface for Object Store v2 data that is external to the Mule application which created or uses the store. An authorized client can call the API for the relevant organization and environment, identify the target Object Store v2 store, and issue a DELETE request for a particular key. This allows Mule application B to remove a value held in Mule application A's Object Store v2 without requiring application A itself to execute the removal operation. The calling application must first obtain appropriate Anypoint Platform authorization and must be granted access to the organization and environment containing the target store. The REST API is specifically intended for operations such as retrieving, storing, and deleting Object Store v2 entries, making it the supported cross-application management mechanism in this scenario. MuleSoft documents the available Object Store v2 API operations, including deletion of stored values, in the Object Store v2 REST API documentation. For the authorization model and platform credentials used to invoke Anypoint Platform APIs, see the Connected Apps overview.

QUESTION NO: 4

An API has been built to enable scheduling email provider. The front-end system does very little data entry validation, and problems have started to appear in the email that go to patients. A validate-customer'' flow is added validate the data.

What is he expected behavior of the 'validate-customer'' flow?

A. If only the email address Is invalid a VALIDATION.INVALID_EMAIL error is raised

B. If the email address is invalid, processing continues to see if the appointment data and customer name are also invalid

C. If the appointment date and customer name are invalid, a SCHEDULE:INVALID_APPOINTMENT_DATE error is raised

D. If all of the values are invalid the last validation error is raised:SCHEDULE:INVALID_CUSTOMER_NAME

ANSWER: A

Explanation:

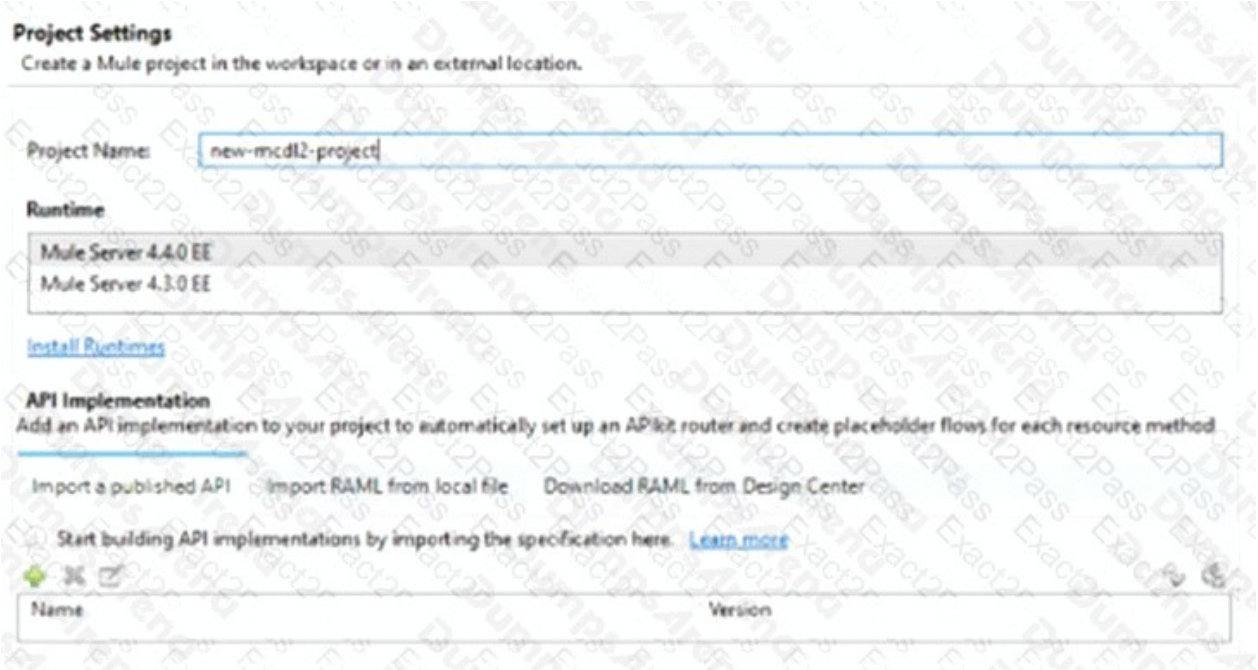
If only the email address Is invalid a VALIDATION.INVALID_EMAIL error is raised is the expected behavior. The validation flow evaluates its message processors in sequence. When the email-validation processor detects an invalid email address, it raises the configured `VALIDATION.INVALID_EMAIL` error. That error interrupts normal execution of the current route, so validation does not continue as though the failed email check had succeeded.

If the validations are placed in an Until Successful scope, Mule retries the scope when an error is thrown, up to the configured retry limit. Each retry starts the scope again; it does not proceed past the failing validation processor to accumulate later validation failures. If the email value remains invalid after the retries are exhausted, the scope propagates the email-validation error to the flow's error handling logic or caller. This behavior ensures that the reported failure accurately identifies the validation condition that prevented successful processing.

MuleSoft documents the retry and error-propagation behavior of Until Successful in the Until Successful Scope documentation. The error type is part of the Validation module's documented validation-error model: Validation Connector documentation.

QUESTION NO: 5

Refer to the exhibit.



When creating a new project, which API implementation allows for selecting the correct API version and scaffolding the flows from the API specification?

A. Import a published API

B. Generate a local RAML from anypoint Studio

C. Download RAML from Design Center

D. Import RAML from local file

ANSWER: A

Explanation:

Import a published API is the correct choice because it uses an API specification published to Anypoint Exchange or Design Center as the source for a new Mule project. During this workflow, Anypoint Studio presents the available published assets and their versions, allowing the developer to select the exact API version that the implementation must support. Studio can then use APIkit to generate the initial implementation structure directly from that selected specification.

The generated project includes APIkit configuration and scaffolded flows for the operations and resources defined in the RAML or OAS contract. This gives the implementation a contract-first starting point, helping ensure the Mule application aligns with the versioned API definition that was designed, reviewed, and published for reuse. Published API import is therefore the appropriate approach when both version selection and flow scaffolding are required as part of project creation. MuleSoft documents this APIkit project-creation workflow at Create an APIkit Project. For the role of Exchange in publishing and consuming versioned API assets, see Anypoint Exchange documentation.

QUESTION NO: 6

The HTTP Request operation raises an HTTP CONNECTIVITY error.

Which HTTP status code and body are returned to the web client?

A. HTTP Status Code:200.Body ‘Error in processing your request

B. HTTP Status Code:500.Body ‘The HTTP CONNECTIVITY Error description

C. HTTP Status Code:500.Body ‘Error in processing your request

D. HTTP Status Code:500.Body ‘Error in processing your request

ANSWER: A

Explanation:

HTTP Status Code:200. Body ‘Error in processing your request is correct because an `on-error-continue` handler handles the `HTTP:CONNECTIVITY` error rather than propagating it out of the flow. The handler explicitly replaces the message payload with `Error in processing your request`, so that value becomes the response body returned by the HTTP Listener.

After an error is handled with `on-error-continue`, Mule considers the flow execution successfully completed. The HTTP Listener therefore uses its normal response path, whose default response status is `200` when no response status variable or explicit response status code is configured. A server-error status is associated with an error that remains unhandled or is propagated to the listener’s error response path; that is not the outcome when the error handler continues execution. MuleSoft documents that `on-error-continue` handles the error and allows the owner flow to complete successfully. See the Mule error handling documentation and the HTTP Listener reference.

QUESTION NO: 7

A Mule application for processing orders must log the order ID for every log message output.

What is a best practice to enrich every log message with the order ID?

A. Use flow variables within every logger processor to log the order ID

B. Set a flow variable and edit the log4/2.xml file to output the variable as part of the message pattern

C. Create a custom XML SDK component to wrap the logger processor and automatically add the order ID within the connector

D. Use the Tracing module to set logging variables with a Mapped Diagnostic Context

ANSWER: D

Explanation:

Use the Tracing module to set logging variables with a Mapped Diagnostic Context is the recommended approach because it adds contextual key-value data, such as an order ID, to the logging context for the current Mule event. The Tracing module’s Set Logging Variables operation is designed specifically to enrich log records consistently without requiring each Logger component to manually construct or repeat the order identifier in its message. After the order ID is set as a logging variable, the application’s Log4j2 layout can include that MDC value in the configured output pattern, allowing log statements produced during processing of the event to be correlated with the relevant order. This provides a clean separation between business-flow logic and logging configuration, improves observability, and makes operational searches and troubleshooting easier when many orders are processed concurrently. The order ID should be set as early as practical in the flow, once it is available, so downstream processing logs carry the same correlation context. MuleSoft documents the Set Logging Variables operation and its use for adding custom logging context in the Tracing Module Reference. For configuration of log output and patterns, see the Mule Runtime Logging documentation.

QUESTION NO: 8

In a Mule project, Flow-1 contains a flow-ref to Flow-2 depends on data from Flow-1 to execute successfully.

Which action ensures the test suites and test cases written for Flow-1 and Flow-2 will executesuccessfully?

A. Chain together the test suites and test cases for Flow-1 and Flow-2

B. Use ‘’Set Event to pass the input that is needed, and keep the test cases for Flow-1 and Flow-2 independent

C. Use ‘’Before Test Case’’ To collect data from Flow-1 test cases before running Flow-2 test cases

D. Use ‘After Test Case’ to produce the data needed from Flow-1 test cases to pass to Flow-2 test cases

ANSWER: B

Explanation:

Use ‘’Set Event to pass the input that is needed, and keep the test cases for Flow-1 and Flow-2 independent is correct because MUnit tests should establish all event data required by the flow under test rather than rely on results produced by a separate test. A flow reference executes the referenced flow using the current Mule event, so Flow-2 requires an event whose payload, attributes, and variables match the data contract supplied by Flow-1 in production. For a direct Flow-2 test, the MUnit Set Event processor can construct that required input explicitly before the flow is invoked. This makes the Flow-2 test deterministic and focused on Flow-2’s behavior while allowing Flow-1’s test to validate its own orchestration and flow-reference behavior independently. Independent tests can run in any order, individually, or as part of a full suite without hidden dependencies on state or execution output from another test case. MUnit documents Set Event as the processor used to define the Mule event data for a test, including payload, variables, and attributes. See MUnit Set Event and MUnit tests.