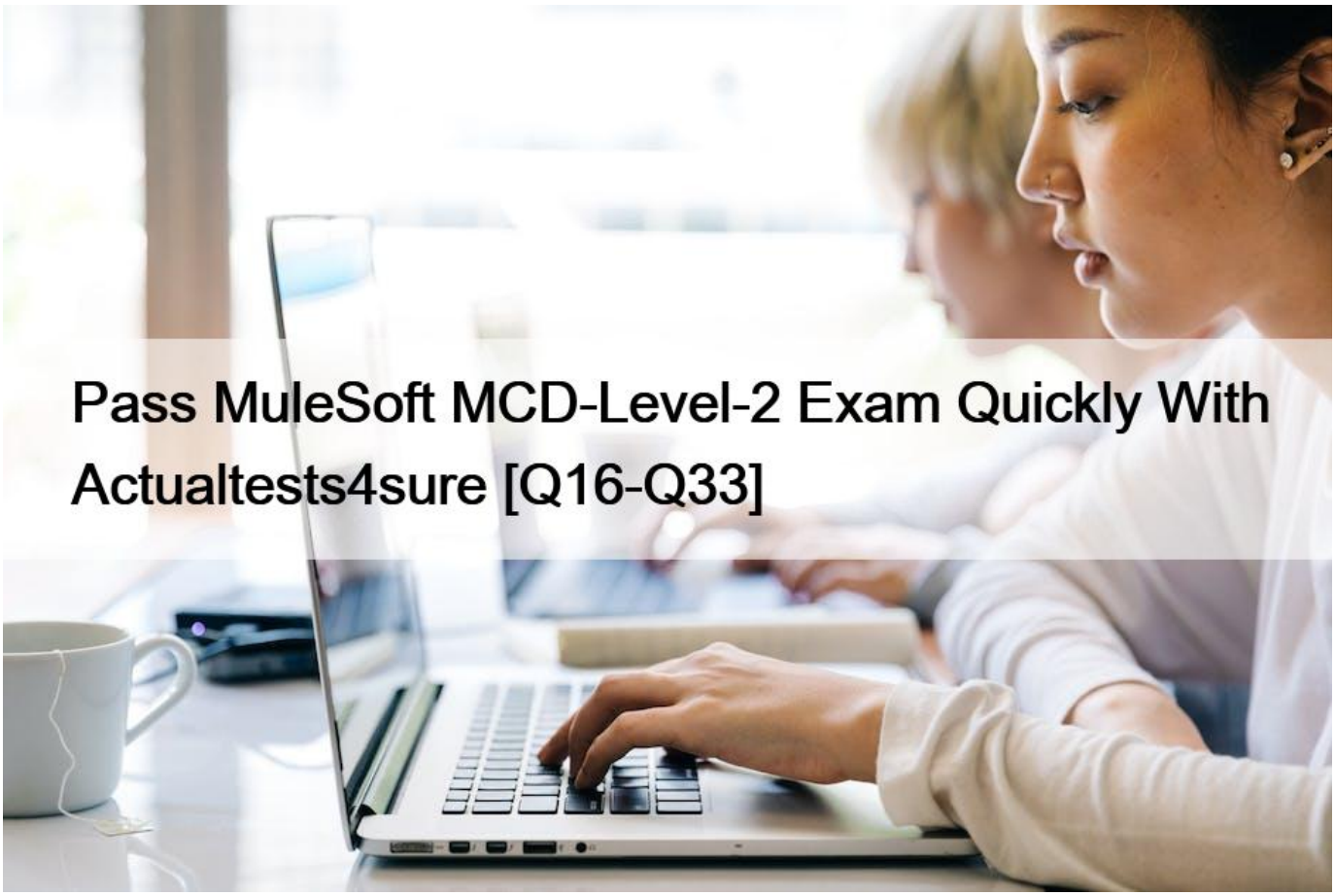
Pass MuleSoft MCD-Level-2 Exam Quickly With Actualtests4sure [Q16-Q33



Pass MuleSoft MCD-Level-2 Exam Quickly With Actualtests4sure
Prepare MCD-Level-2 Question Answers - MCD-Level-2 Exam Dumps

QUESTION 16

When registering a client application with an existing API instance or API Group instance, what is required to manually approve or reject request access?
* To configure the SLA tier for the application and have the role of Organization Administrator, API Manager Environment Administrator, or the Manage Contacts permission
* To configure the SLA tier for the application and have the Exchange Administrator permission
* To configure the SLA tier for the application
* To only have Exchange Administrator permission
Explanation

To manually approve or reject request access when registering a client application with an existing API instance or API Group instance, it is required to configure the SLA tier for the application and have one of the following roles or permissions: Organization Administrator, API Manager Environment Administrator, or Manage Contracts permission. These roles or permissions allow managing client applications and contracts in API Manager. References:

https://docs.mulesoft.com/api-manager/2.x/client-applications#managing-client-applications-and-contracts

QUESTION 17

Which statement is true when using XML SDK for creating custom message processors?
* Properties are fields defined by an end user of the XML SDK component and serve as a global configuration for the entire Mule project in which they are used
* An XML SDK provides both inbound and outbound operations
* Operations can be reused in recursive calls
* All operations are public
Explanation

When using XML SDK for creating custom message processors, all operations are public by default and can be used by any Mule application that imports them. There is no way to make an operation private or protected in XML SDK. References: https://docs.mulesoft.com/mule-sdk/1.1/xml-sdk#operations

QUESTION 18

An order processing system is composed of multiple Mule application responsible for warehouse, sales and shipping. Each application communication using Anypoint MQ. Each message must be correlated against the original order ID for observability and tracing.

How should a developer propagate the order ID as the correlation ID across each message?
* Use the underlying HTTP request of Anypoint MQ to set the ‘X-CORRELATION_ID’ header to the order ID
* Set a custom Anypoint MQ user property to propagate the order ID and set the correlation ID in the receiving applications.
* Use the default correlation ID, Anypoint MQ will sutomatically propagate it.
* Wrap all Anypoint MQ Publish operations within a With CorrelationID scope from the Tracing module, setting the correlation ID to the order ID
Explanation

To propagate the order ID as the correlation ID across each message using Anypoint MQ, the developer should wrap all Anypoint MQ Publish operations within a With CorrelationID scope from the Tracing module, setting the correlation ID to the order ID. The With CorrelationID scope allows setting a custom correlation ID for any event that occurs within it. The Tracing module also enables distributed tracing across different Mule applications and services using Anypoint Monitoring. References:

https://docs.mulesoft.com/tracing-module/1.0/tracing-module-reference#with-correlation-id-scope

https://docs.mulesoft.com/tracing-module/1.0/tracing-module-concepts

QUESTION 19

A Mule API receives a JSON payload and updates the target system with the payload. The developer uses JSON schemas to ensure the data is valid.

How can the data be validation before posting to the target system?
* Use a DataWeave 2.09 transform operation, and at the log of the DataWeave script, add:

%dw 2.0

Import.json-moduls
* Using the DataWeave if Else condition test the values of the payload against the examples included in the schema
* Apply the JSON Schema policy in API Manager and reference the correct schema in the policy configuration
* Add the JSON module dependency and add the validate-schema operation in the flow, configured to reference the schema

Explanation

To validate the data before posting to the target system, the developer should add the JSON module dependency and add the validate-schema operation in the flow, configured to reference the schema. The JSON module provides a validate-schema operation that validates a JSON payload against a JSON schema and throws an error if the payload is invalid. References:

https://docs.mulesoft.com/json-module/1.1/json-validate-schema

QUESTION 20

A Mule application deployed to a standardalone Mule runtime uses VM queues to publish messages to be consumed asynchronously by another flow.

In the case of a system failure, what will happen to in-flight messages in the VM queues that have been consumed?
* For nay type of queue, the message will be processed after the system comes online
* For persistent queues, the message will be processed after the system comes online
* For transient queues, the message will be processed after the system comes online
* For any type of queue, the message will be lost

Explanation

In case of a system failure, in-flight messages in persistent VM queues that have been consumed will be processed after the system comes online. This is because persistent VM queues store messages on disk and guarantee delivery even if there is a system crash or restart. Therefore, any in-flight messages that have been consumed but not processed will be recovered from disk and processed when the system is back online.

References: https://docs.mulesoft.com/mule-runtime/4.3/vm-connector#persistent-queues

QUESTION 21

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?
* Use flow variables within every logger processor to log the order ID
* Set a flow variable and edit the log4/2.xml file to output the variable as part of the message pattern
* Create a custom XML SDK component to wrap the logger processor and automatically add the order ID within the connector
* Use the Tracing module to set logging variables with a Mapped Diagnostic Context

Explanation

To enrich every log message with the order ID, the developer should use the Tracing module to set logging variables with a Mapped Diagnostic Context (MDC). The Tracing module allows adding custom key-value pairs to log messages using MDC variables. The developer can use Set Logging Variables operation to set the order ID as an MDC variable and then use it in any logger processor within the same thread or event.

References: https://docs.mulesoft.com/tracing-module/1.0/tracing-module-reference#set-logging-variables

QUESTION 22

A Flight Management System publishes gate change notification events whenever a flight’s arrival gate changes. Other systems, including Baggage Handler System. Inflight Catering System and Passenger Notifications System, must each asynchronously receive the same gate change notification to process the event according.

Which configuration is required in Anypoint MQ to achieve this publish/subscribe model?
* Publish each client subscribe directly to the exchange.

Have each client subscribe directly to the queue.
* Publish the gate change notification to an Anypoint MC queue

Have each client subscribe directly to the queue
* Publish the gate change notification to an Anypoint MQ queue.

Create different anypoint MQ exchange meant for each of the other subscribing systems Bind the queue with each of the exchanges
* Publish the gate change notification to an Anypoint MQ exchanhe.

Create different Anypoint MQ queues meant for each of the other subscribing systems.

Bind the exchange with each of the queues.
Explanation

To achieve a publish/subscribe model using Anypoint MQ, where each system receives the same gate change notification event, the developer should publish the gate change notification to an Anypoint MQ exchange, create different Anypoint MQ queues meant for each of the other subscribing systems, and bind the exchange with each of the queues. An exchange is a message routing agent that can send messages to different queues based on predefined criteria. By binding an exchange with multiple queues, each queue receives a copy of every message sent to that exchange. Therefore, each system can subscribe to its own queue and receive every gate change notification event. References:

https://docs.mulesoft.com/anypoint-mq/3.x/anypoint-mq-exchanges

QUESTION 23

An API has been developed and deployed to CloudHub Among the policies applied to this API is an allowlist of IP addresses. A developer wants to run a test in Anypoint Studio and does not want any policies applied because their workstation is not included in the allowlist.

What must the developer do in order to run this test locally without the policies applied?
* Create a properties file specifically for local development and set the API instance ID to a value that is not used in API Manager
* Pass in the runtime parameter ”-Danpow.platform.gatekeeper=disabled”
* Deactivate the API in API Manager so the Autodiscovery element will not find the application when it runs in Studio
* Run the test as-s, with no changes because the Studio runtime will not attempt to connect to API Manager
Explanation

To run a test locally without the policies applied, the developer should create a properties file specifically for local development and set the API instance ID to a value that is not used in API Manager. This way, the developer can use different configuration properties for different environments and avoid triggering API autodiscovery when running tests locally. API autodiscovery is a mechanism that associates an API implementation with its corresponding API specification and policies in API Manager based on its API instance ID. By setting this ID to a value that does not exist in API Manager, the developer can prevent API autodiscovery from finding and applying any policies to the local test. References:

https://docs.mulesoft.com/api-manager/2.x/api-auto-discovery-new-concept#configuring-api-autodiscovery

https://docs.mulesoft.com/mule-runtime/4.3/configuring-properties

QUESTION 24

The flow is invoicing a target API. The API’s protocol is HTTPS. The TLS configuration in the HTTP Request Configuration global element is set to None. A web client submits a request to

http:localhost:8081/vehicles.

If the certificate of the target API is signed by a certificate authority (CA), what is true about the HTTP Request operation when the flow executes?

* The HTTP Request operation will succeed if the CA’S certificate is present in the JRE’s default keystore
* The HTTP Request operation will succeed if the CA’s certificate is present in the JRE’s default truststore.
* The HTTP Request operation will always succeed regardless of the CA
* The HTTP Request operation will always fail regardless of the CA

Explanation

The HTTP Request operation will use the default truststore of the JRE to validate the certificate of the target API. If the CA’s certificate is present in the truststore, the operation will succeed. Otherwise, it will fail with a handshake exception. References: https://docs.mulesoft.com/mule-runtime/4.3/tls-configuration#tls-default

QUESTION 25

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?

* Create a health check endpoint that listens on a separate port and uses a separate HTTP Listener configuration from the API
* Configure the application to send health data to an external system
* Create a health check endpoint that reuses the same port number and HTTP Listener configuration as the API itself
* Monitor the Payment API directly sending real customer payment data

Explanation

To test that the payment API is working in production, the developer should create a health check endpoint that listens on a separate port and uses a separate HTTP Listener configuration from the API. This way, the developer can isolate the health check endpoint from the API traffic and avoid affecting the performance or availability of the API. The health check endpoint should return a simple response that indicates the status of the API, such as OK or ERROR. References:

https://docs.mulesoft.com/api-functional-monitoring/afm-create-monitor#create-a-monitor

QUESTION 26

Refer to the exhibit.

```
<flow name="implementation" >
    <raise-error doc:name="Raise error" type="APP:CUSTOM_ERROR"/>
</flow>
```

```
<munit:test name="start-up-test" description="Test that Mule app starts up" expectedErrorType="
    <munit:execution>
        <flow-ref doc:name="implementation" name="implementation"/>
    </munit:execution>
    <munit:validation >
        <munit-tools:assert-that doc:name="Assert that" expression="#[true]" is="#[MunitTools:
    </munit:validation>
</munit:test>
```


The flow name is ”implementation” with code for the MUnit test case.

When the MUnit test case is executed, what is the expected result?
* The test case fails with an assertion error
* The test throws an error and does not start
* The test case fails with an unexpected error type
* The test case passes
Explanation

Based on the code snippet and MUnit test case below, when the MUnit test case is executed, the expected result is that the test case fails with an assertion error. This is because the assert-equals processor compares two values for equality, and fails if they are not equal. In this case, the expected value is ‘Hello World’, but the actual value returned by the implementation flow is ‘Hello Mule’. Therefore, the assertion fails and an error is thrown. References:
https://docs.mulesoft.com/munit/2.3/assert-equals-processor

QUESTION 27

What is the MuleSoft recommended method to encrypt sensitive property data?
* The encryption key and sensitive data should be different for each environment
* The encryption key should be identical for all environments
* The encryption key should be identical for all environments and the sensitive data should be different for each environment
* The encryption key should be different for each environment and the sensitive data should be the same for all environments
Explanation

The MuleSoft recommended method to encrypt sensitive property data is to use the Secure Properties Tool that comes with Anypoint Studio. This tool allows encrypting properties files with a secret key and then decrypting them at runtime using the same key. The encryption key and sensitive data should be different for each environment to ensure security and avoid accidental exposure of sensitive data. References:

https://docs.mulesoft.com/mule-runtime/4.3/secure-configuration-properties

QUESTION 28

A system API that communicates to an underlying MySQL database is deploying to CloudHub. The DevOps team requires a readiness endpoint to monitor all system APIs.

Which strategy should be used to implement this endpoint?
* Create a dedicated endpoint that responds with the API status and reachability of the underlying systems
* Create a dedicated endpoint that responds with the API status and health of the server
* Use an existing resource endpoint of the API
* Create a dedicated endpoint that responds with the API status only
Explanation

To implement a readiness endpoint to monitor all system APIs, the developer should create a dedicated endpoint that responds with the API status and reachability of the underlying systems. This way, the DevOps team can check if the system API is ready to receive requests and if it can communicate with its backend systems without errors. References:

https://docs.mulesoft.com/mule-runtime/4.3/deployment-strategies#readiness-probes

QUESTION 29

Which properties are mandatory on the HTTP Connector configuration in order to use the OAuth 2.0 Authorization Code grant type

for authentication?
* External callback URL, access token URL, client ID response access token
* Token URL, authorization URL, client ID, client secret local callback URL
* External callback URL, access token URL, client ID, response refresh token
* External callback URL, access token URL, local authorization URL, authorization URL, client ID, client secret
Explanation

To use the OAuth 2.0 Authorization Code grant type for authentication, the HTTP Connector configuration requires the following properties: token URL, authorization URL, client ID, client secret, and local callback URL. The token URL is the endpoint of the authorization server that provides access tokens. The authorization URL is the endpoint of the authorization server that initiates the user consent flow. The client ID and client secret are the credentials of the Mule application registered with the authorization server. The local callback URL is the endpoint of the Mule application that receives the authorization code from the authorization server.

References: https://docs.mulesoft.com/http-connector/1.6/http-authentication#oauth-2-0

QUESTION 30

An organization uses CloudHub to deploy all of its applications.

How can a common-global-handler flow be configured so that it can be reused across all of the organization’s deployed applications?
* Create a Mule plugin project

Create a common-global-error-handler flow inside the plugin project.

Use this plugin as a dependency in all Mute applications.

Import that configuration file in Mute applications.
* Create a common-global-error-handler flow in all Mule Applications Refer to it flow-ref wherever needed.
* Create a Mule Plugin project

Create a common-global-error-handler flow inside the plugin project.

Use this plugin as a dependency in all Mule applications
* Create a Mule daman project.

Create a common-global-error-handler flow inside the domain project.

Use this domain project as a dependency.
Explanation

To configure a common-global-handler flow that can be reused across all of the organization’s deployed applications, the developer should create a Mule Plugin project, create a common-global-error-handler flow inside the plugin project, and use this plugin as a dependency in all Mule applications. This way, the developer can import the common-global-error-handler flow in any application that needs it and avoid duplicating the error handling logic. References:

https://docs.mulesoft.com/mule-runtime/4.3/error-handling#global-error-handler

QUESTION 31

A Mule implementation uses a HTTP Request within an Unit Successful scope to connect to an API.

How should a permanent error response like HTTP:UNAUTHORIZED be handle inside Until Successful to reduce latency?
* Configure retrying until a MULERETRY_EXHAUSTED error is raised or the API responds back with a successful response.
* In Until Successful configuration, set the retry count to 1 for error type HTTP: UNAUTHORIZED.
* Put the HTTP Request inside a try scope in Unit Successful.

In the error handler, use On Error Continue to catch permanent errors like HTTP UNAUTHORIZED.
* Put the HTTP Request inside a try scope in Unit Successful.

In the error handler, use On Error Propagate to catch permanent errors like HTTP UNAUTHORIZED.
Explanation

To handle a permanent error response like HTTP:UNAUTHORIZED inside Until Successful, the developer should put the HTTP Request inside a try scope in Unit Successful, and use On Error Continue to catch permanent errors like HTTP UNAUTHORIZED in the error handler. This way, the developer can avoid retrying requests that will always fail due to a permanent error, and reduce latency. On Error Continue allows the flow to continue processing after handling the error. References:

https://docs.mulesoft.com/mule-runtime/4.3/until-successful-scope

https://docs.mulesoft.com/mule-runtime/4.3/on-error-continue-concept

QUESTION 32

The Center for Enablement team published a common application as a reusable module to the central Nexus repository.

How can the common application be included in all API implementations?
* Download the common application from Naxus and copy it to the src/main/resources folder in the API
* Copy the common application’s source XML file and out it in a new flow file in the src/main/mule folder
* Add a Maven dependency in the PCM file with multiple-plugin as <classifier>
* Add a Maven dependency in the POM file with jar as <classifier>
Explanation

To include a common application as a reusable module in all API implementations, the developer should add a Maven dependency in the POM file with jar as <classifier>. This way, the developer can reuse Mule code from another application by packaging it as a JAR file and adding it as a dependency in the POM file of the API implementation. The classifier element specifies that it is a JAR file. References:

https://docs.mulesoft.com/mule-runtime/4.3/mmp-concept#add-a-maven-dependency-to-the-pom-file

QUESTION 33

Which command is used to convert a JKS keystore to PKCS12?
* Keytool-importkeystore -srckeystore keystore p12-srcstoretype PKCS12 -destkeystore keystore.jks

-deststoretype JKS
* Keytool-importkeystore -srckeystore keystore p12-srcstoretype JKS -destkeystore keystore.p12

-deststoretype PKCS12
* Keytool-importkeystore -srckeystore keystore jks-srcstoretype JKS -destkeystore keystore.p13

-deststoretype PKCS12
* Keytool-importkeystore -srckeystore keystore jks-srcstoretype PKCS12 -destkeystore keystore.p12

-deststoretype JKS
Explanation

To convert a JKS keystore to PKCS12, the developer needs to use the keytool-importkeystore command with the following options: -srckeystore keystore.jks -srcstoretype JKS -destkeystore keystore.p12 -deststoretype PKCS12. This command imports all entries from a source JKS keystore (keystore.jks) into a destination PKCS12 keystore (keystore.p12). References:

https://docs.oracle.com/en/java/javase/11/tools/keytool.html#GUID-5990A2E4-78E3-47B7-AE75-6D182625954