MCPA-LEVEL1 Q&As

MuleSoft Certified Platform Architect - Level 1

Pass Mulesoft MCPA-LEVEL1 Exam with 100% Guarantee

Free Download Real Questions & Answers **PDF** and **VCE** file from:

https://www.geekcert.com/mulesoft-certified-platform-architect-level-1.html

100% Passing Guarantee
100% Money Back Assurance

Following Questions and Answers are all new published by Mulesoft Official Exam Center

- **Instant Download** After Purchase
- **100% Money Back** Guarantee
- **365 Days** Free Update
- **800,000+** Satisfied Customers

QUESTION 1

A REST API is being designed to implement a Mule application.

What standard interface definition language can be used to define REST APIs?

A. Web Service Definition Language(WSDL)

B. OpenAPI Specification (OAS)

C. YAML

D. AsyncAPI Specification

Correct Answer: B

QUESTION 2

A system API is deployed to a primary environment as well as to a disaster recovery (DR) environment, with different DNS names in each environment. A process API is a client to the system API and is being rate limited by the system API, with different limits in each of the environments. The system API\\'s DR environment provides only 20% of the rate limiting offered by the primary environment. What is the best API fault-tolerant invocation strategy to reduce overall errors in the process API, given these conditions and constraints?

A. Invoke the system API deployed to the primary environment; add timeout and retry logic to the process API to avoid intermittent failures; if it still fails, invoke the system API deployed to the DR environment

B. Invoke the system API deployed to the primary environment; add retry logic to the process API to handle intermittent failures by invoking the system API deployed to the DR environment

C. In parallel, invoke the system API deployed to the primary environment and the system API deployed to the DR environment; add timeout and retry logic to the process API to avoid intermittent failures; add logic to the process API to combine the results

D. Invoke the system API deployed to the primary environment; add timeout and retry logic to the process API to avoid intermittent failures; if it still fails, invoke a copy of the process API deployed to the DR environment

Correct Answer: A

Invoke the system API deployed to the primary environment; add timeout and retry logic to the process API to avoid intermittent failures; if it still fails, invoke the system API deployed to the DR environment
********** There is one important consideration to be noted in the question which is - System API in DR environment provides only 20% of the rate limiting offered by the primary environment. So, comparitively, very less calls will be allowed into the DR environment API opposed to its primary environment. With this in mind, lets analyse what is the right and best fault- tolerant invocation strategy.

1.

Invoking both the system APIs in parallel is definitely NOT a feasible approach because of the 20% limitation we have on DR environment. Calling in parallel every time would easily and quickly exhaust the rate limits on DR environment and may not give chance to genuine intermittent error scenarios to let in during the time of need.

2.

Another option given is suggesting to add timeout and retry logic to process API while invoking primary environment\\'s system API. This is good so far. However, when all retries failed, the option is suggesting to invoke the copy of process API on DR environment which is not right or recommended. Only system API is the one to be considered for fallback and not the whole process API. Process APIs usually have lot of heavy orchestration calling many other APIs which we do not want to repeat again by calling DR\\'s process API. So this option is NOT right.

3.

One more option given is suggesting to add the retry (no timeout) logic to process API to directly retry on DR environment\\'s system API instead of retrying the primary environment system API first. This is not at all a proper fallback. A proper fallback should occur only after all retries are performed and exhausted on Primary environment first. But here, the option is suggesting to directly retry fallback API on first failure itself without trying main API. So, this option is NOT right too.

This leaves us one option which is right and best fit.

-Invoke the system API deployed to the primary environment

-Add Timeout and Retry logic on it in process API

- If it fails even after all retries, then invoke the system API deployed to the DR environment.

QUESTION 3

An API implementation is being designed that must invoke an Order API, which is known to repeatedly experience downtime.

For this reason, a fallback API is to be called when the Order API is unavailable.

What approach to designing the invocation of the fallback API provides the best resilience?

A. Search Anypoint Exchange for a suitable existing fallback API, and then implement invocations to this fallback API in addition to the Order API

B. Create a separate entry for the Order API in API Manager, and then invoke this API as a fallback API if the primary Order API is unavailable

C. Redirect client requests through an HTTP 307 Temporary Redirect status code to the fallback API whenever the Order API is unavailable

D. Set an option in the HTTP Requester component that invokes the Order API to instead invoke a fallback API whenever an HTTP 4xx or 5xx response status code is returned from the Order API

Correct Answer: A

Search Anypoint exchange for a suitable existing fallback API, and then implement invocations to this fallback API in addition to the order API ********** >> It is not ideal and good approach, until unless there is a pre-approved agreement with the API clients that they will receive a HTTP 3xx temporary redirect status code and they have to implement fallback logic their side to call another API. >> Creating separate entry of same Order API in API manager would just create an another instance of it on top of same API implementation. So, it does NO GOOD by using clone od same API as a fallback API. Fallback API should be ideally a different API implementation that is not same as primary one. >> There is NO option currently provided by Anypoint HTTP Connector that allows us to invoke a fallback API when we receive certain HTTP status codes in response. The only statement TRUE in the given options is to Search Anypoint exchange for a suitable existing fallback API, and then implement invocations to this fallback API in addition to the order API.

QUESTION 4

A set of tests must be performed prior to deploying API implementations to a staging environment. Due to data security and access restrictions, untested APIs cannot be granted access to the backend systems, so instead mocked data must be used for these tests. The amount of available mocked data and its contents is sufficient to entirely test the API implementations with no active connections to the backend systems. What type of tests should be used to incorporate this mocked data?

A. Integration tests

B. Performance tests

C. Functional tests (Blackbox)

D. Unit tests (Whitebox)

Correct Answer: D

Unit tests (Whitebox)

****

Reference: https://docs.mulesoft.com/mule-runtime/3.9/testing-strategies

As per general IT testing practice and MuleSoft recommended practice, Integration and Performance tests should be done on full end to end setup for right evaluation. Which means all end systems should be connected while doing the tests.

So, these options are OUT and we are left with Unit Tests and Functional Tests. As per attached reference documentation from MuleSoft:

Unit Tests - are limited to the code that can be realistically exercised without the need to run it inside Mule itself. So good candidates are Small pieces of modular code, Sub Flows, Custom transformers, Custom components, Custom

expression evaluators etc.

Functional Tests - are those that most extensively exercise your application configuration. In these tests, you have the freedom and tools for simulating happy and unhappy paths. You also have the possibility to create stubs for target services

and make them success or fail to easily simulate happy and unhappy paths respectively.

As the scenario in the question demands for API implementation to be tested before deployment to Staging and also clearly indicates that there is enough/ sufficient amount of mock data to test the various components of API implementations

with no active connections to the backend systems, Unit Tests are the one to be used to incorporate this mocked data.

QUESTION 5

An organization wants to make sure only known partners can invoke the organization\\'s APIs. To achieve this security goal, the organization wants to enforce a Client ID Enforcement policy in API Manager so that only registered partner applications can invoke the organization\\'s APIs. In what type of API implementation does MuleSoft recommend adding

an API proxy to enforce the Client ID Enforcement policy, rather than embedding the policy directly in the application\\'s JVM?

A. A Mule 3 application using APIkit

B. A Mule 3 or Mule 4 application modified with custom Java code

C. A Mule 4 application with an API specification

D. A Non-Mule application

Correct Answer: D

A Non-Mule application

****

>> All type of Mule applications (Mule 3/ Mule 4/ with APIkit/ with Custom Java Code etc) running on Mule Runtimes support the Embedded Policy Enforcement on them. >> The only option that cannot have or does not support embedded

policy enforcement and must have API Proxy is for Non-Mule Applications.

So, Non-Mule application is the right answer.