PassQueen
a.k.a dumpsqueen.com

MuleSoft Certified Platform Architect - Level 1 MAINTENANCE

Mulesoft MCPA-Level-1-Maintenance

Version Demo

Total Demo Questions: 11

Total Premium Questions: 110

Buy Premium PDF

https://passqueen.com

support@passqueen.com

QUESTION NO: 1

A Mule application receives order requests and publishes each order to Anypoint MQ for asynchronous processing. The order-processing consumer application must ensure that a message is not permanently removed from the queue until processing has completed successfully.

What action should the consumer application take after successful processing?

A. Acknowledge the message after successful processing

B. Delete the queue after successful processing

C. Republish the message to the same queue after successful processing

D. Stop the subscriber after successful processing

ANSWER: A

Explanation:

Acknowledge the message after successful processing is correct because message acknowledgment confirms to Anypoint MQ that the consumer has completed handling the message. The queue can then remove that message rather than making it available for redelivery.

If processing fails before acknowledgment, the message can remain eligible for redelivery according to the queue and subscriber configuration. This supports at-least-once delivery patterns, so the consuming application should also be designed to tolerate duplicate deliveries where required. Acknowledging only after successful processing helps avoid losing an order merely because the consumer failed during processing. See Anypoint MQ documentation.

QUESTION NO: 2

What is true about where an API policy is defined in Anypoint Platform and how it is then applied to API instances?

A. The API policy Is defined In Runtime Manager as part of the API deployment to a Mule runtime, and then ONLY applied to the specific API Instance

B. The API policy Is defined In API Manager for a specific API Instance, and then ONLY applied to the specific API instance

C. The API policy Is defined in API Manager and then automatically applied to ALL API instances

D. The API policy is defined in API Manager, and then applied to ALL API instances in the specified environment

ANSWER: B

Explanation:

The API policy Is defined In API Manager for a specific API Instance, and then ONLY applied to the specific API instance. In Anypoint Platform, policies are managed through API Manager and are attached to an API instance, rather than being defined as part of deploying an application in Runtime Manager. An API instance represents a particular managed endpoint and deployment context for an API, such as a development, test, or production exposure. This instance-level association lets an organization apply the controls needed for that particular endpoint, including security, traffic management, transformation, and governance policies.

API Manager supports managing multiple instances of the same API independently. Consequently, each instance can have its own policy configuration according to its operational and nonfunctional requirements. Policies are enforced at runtime by the Mule Gateway capability associated with the managed API, but their lifecycle and configuration remain in API Manager. This separation enables platform teams to govern API access and behavior without changing the underlying API implementation for every policy update. MuleSoft documents API Manager as the central service for managing API instances and applying policies; see the API Manager documentation and the API instance management guide.

QUESTION NO: 3

What is true about API implementations when dealing with legal regulations that require all data processing to be performed within a certain jurisdiction (such as in the USA or the EU)?

A. They must avoid using the Object Store as it depends on services deployed ONLY to the US East region

B. They must use a Jurisdiction-local external messaging system such as Active MQ rather than Anypoint MQ

C. They must be deployed to Anypoint Platform runtime planes that are managed by Anypoint Platform control planes, with both planes in the same Jurisdiction

D. They must ensure ALL data is encrypted both in transit and at rest

ANSWER: C

Explanation:

They must be deployed to Anypoint Platform runtime planes that are managed by Anypoint Platform control planes, with both planes in the same jurisdiction. Data-residency obligations apply to more than the business payload handled by an API. Depending on the deployment model and enabled platform capabilities, operational information can include application configuration, deployment metadata, logs, monitoring and analytics data, and state used by platform services. Consequently, an architecture intended to meet a jurisdictional processing requirement must account for the location of both the runtime that executes the API and the Anypoint Platform control plane that manages it.

Anypoint Platform provides separate regional control planes, including US and EU regions. Selecting the appropriate regional control plane and using runtime infrastructure located in the corresponding jurisdiction keeps management and runtime processing aligned with the applicable residency boundary. This is an architectural deployment decision that should be made early, because the organization and its associated platform data are tied to the selected control-plane region. MuleSoft documents regional organizations and their regional service endpoints in its organization documentation, and describes how Runtime Manager deploys and manages applications in the Runtime Manager documentation. The resulting design provides the required jurisdictional alignment for the API implementation and its supporting platform operations.

QUESTION NO: 4

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

ANSWER: A

Explanation:

Search Anypoint Exchange for a suitable existing fallback API, and then implement invocations to this fallback API in addition to the Order API is the most resilient approach because it uses a separately implemented, discoverable service as an alternate dependency. Anypoint Exchange is MuleSoft's catalog for reusable APIs, assets, and connectors, so it is the appropriate place to identify an approved API that can provide the required fallback capability. Reusing a governed asset

also helps maintain consistent contracts, security expectations, ownership, and lifecycle management across the integration landscape.

The implementation should invoke the Order API as the primary path and handle connectivity or availability failures in the Mule flow. In the relevant error-handling path, it can invoke the selected fallback API and return or transform the fallback response according to the implementation's contract. This design contains the outage-handling behavior within the integration layer, where retry, timeout, error mapping, logging, and observability can be consistently managed. Mule error handling supports continuing processing through an error handler after a failure, enabling the flow to execute an alternate recovery action such as calling the fallback service.

See MuleSoft's documentation for Anypoint Exchange and Mule error handling.

QUESTION NO: 5

What Anypoint Platform Capabilities listed below fall under APIs and API Invocations/Consumers category? Select TWO.

A. API Operations and Management

B. API Runtime Execution and Hosting

C. API Consumer Engagement

D. API Design and Development

ANSWER: D

Explanation:

API Design and Development is the correct selection because it is a core Anypoint Platform capability for defining, documenting, testing, and implementing APIs before they are deployed and governed. MuleSoft's API-led approach begins with creating a clear API contract, commonly using RAML or OAS, so that an API can be reviewed, discovered, and reused consistently across teams. Anypoint Design Center supports the design of API specifications, while Anypoint Studio enables developers to implement API interfaces and integrations as Mule applications. These activities establish the API's intended interface and behavior, supporting a design-first workflow in which consumers and implementation teams can work from a shared contract. MuleSoft documents API design as a foundational practice for producing reusable, consistent APIs and provides Design Center as the platform capability for creating and publishing API specifications. See the Anypoint Design Center documentation and MuleSoft's API Designer documentation for details on designing API specifications and managing their lifecycle.

QUESTION NO: 6

Mule applications that implement a number of REST APIs are deployed to their own subnet that is inaccessible from outside the organization.

External business-partners need to access these APIs, which are only allowed to be invoked from a separate subnet dedicated to partners - called Partner-subnet. This subnet is accessible from the public internet, which allows these external partners to reach it.

Anypoint Platform and Mule runtimes are already deployed in Partner-subnet. These Mule runtimes can already access the APIs.

What is the most resource-efficient solution to comply with these requirements, while having the least impact on other applications that are currently using the APIs?

A. Implement (or generate) an API proxy Mule application for each of the APIs, then deploy the API proxies to the Mule runtimes

B. Redeploy the API implementations to the same servers running the Mule runtimes

C. Add an additional endpoint to each API for partner-enablement consumption

D. Duplicate the APIs as Mule applications, then deploy them to the Mule runtimes

ANSWER: A

Explanation:

Implementing (or generating) an API proxy Mule application for each of the APIs, then deploying the API proxies to the Mule runtimes is the most resource-efficient approach because it creates a controlled ingress layer in the Partner-subnet without relocating, modifying, or duplicating the existing API implementations. Each proxy can expose a partner-facing endpoint reachable from the public-facing subnet and route approved requests to the existing API endpoint across the internal network path already available to those runtimes. Existing consumers can continue invoking the original APIs through their current endpoints, so their integrations do not need to change.

An API proxy is purpose-built for this separation of concerns: it provides an independently deployable gateway layer in front of an implementation while preserving the backend API. The proxy can also apply API Manager policies, such as client authentication, rate limiting, and traffic controls, at the partner entry point. This centralizes partner access governance and avoids consuming the additional infrastructure and maintenance capacity that separate implementation copies would require. MuleSoft documents that API proxies are deployed separately from API implementations and are managed through API Manager. See the API proxy documentation and MuleSoft's API management documentation.

QUESTION NO: 7

In which layer of API-led connectivity, does the business logic orchestration reside?

A. System Layer

B. Experience Layer

C. Process Layer

ANSWER: C

Explanation:

Business logic orchestration resides in the **Process Layer**. In MuleSoft's API-led connectivity architecture, process APIs compose and coordinate data and capabilities exposed by one or more system APIs to implement business processes. This is the appropriate layer for applying reusable business logic, such as coordinating customer, order, inventory, or fulfillment operations across multiple underlying systems. Process APIs are intentionally independent of the specific channels that consume them, allowing the same orchestrated business capability to be reused by several experience APIs.

This separation helps organizations avoid duplicating orchestration logic for each consumer channel and keeps direct connectivity to systems of record isolated in system APIs. A process API can retrieve, transform, combine, and route information from multiple systems while presenting a business-oriented service that downstream consumers can use consistently. MuleSoft describes the process layer as the layer that orchestrates and combines system APIs into business processes; see the MuleSoft API-led connectivity documentation and the API-led connectivity overview.

QUESTION NO: 8

What is the main change to the IT operating model that MuleSoft recommends to organizations to improve innovation and clock speed?

A. Drive consumption as much as production of assets; this enables developers to discover and reuse assets from other projects and encourages standardization

B. Expose assets using a Master Data Management (MDM) system; this standardizes projects and enables developers to quickly discover and reuse assets from other projects

C. Implement SOA for reusable APIs to focus on production over consumption; this standardizes on XML and WSDL formats to speed up decision making

D. Create a lean and agile organization that makes many small decisions everyday; this speeds up decision making and enables each line of business to take ownership of its projects

ANSWER: A

Explanation:

Drive consumption as much as production of assets; this enables developers to discover and reuse assets from other projects and encourages standardization is correct because MuleSoft's recommended operating model treats reusable integration assets, APIs, and related resources as products that must be easy for teams across the organization to find, understand, access, and adopt. Improving delivery speed is not only about building assets efficiently; it also depends on enabling other teams to consume existing capabilities rather than repeatedly creating point-to-point integrations or rebuilding the same functionality. This production-and-consumption mindset supports API-led connectivity by organizing capabilities into reusable, discoverable building blocks and making self-service reuse a normal part of software delivery. A Center for Enablement helps establish the standards, governance, platforms, documentation, and enablement practices that make this model practical at scale. As consumption rises, teams can compose solutions from proven assets, reduce duplicate effort, and focus innovation on new business value. MuleSoft describes this shift as moving beyond a traditional project-centric delivery approach toward reusable assets that empower decentralized teams while maintaining organizational standards. This improves both innovation capacity and organizational clock speed. See MuleSoft's overview of the Center for Enablement and its discussion of managing IT projects through reuse.

QUESTION NO: 9

A Process API must retrieve customer data from two independent System APIs before returning a combined response. The calls have no dependency on each other, and reducing the overall response time is important.

What Mule routing capability is most appropriate?

A. Scatter-Gather router

B. Choice router

C. Until Successful scope

D. For Each scope

ANSWER: A

Explanation:

A Scatter-Gather router is appropriate because it sends a Mule event to multiple routes concurrently and then aggregates the route results. When the two System API calls are independent, concurrent execution can reduce the elapsed time compared with invoking the APIs sequentially.

Each route can invoke one System API, and subsequent flow logic can transform the gathered results into the Process API response. Error-handling behavior must also be designed deliberately, because a failure in one route can affect the overall Scatter-Gather result according to the configured application error handling.

See MuleSoft documentation for the Scatter-Gather router.

QUESTION NO: 10

What correctly characterizes unit tests of Mule applications?

A. They test the validity of input and output of source and target systems

B. They must be run in a unit testing environment with dedicated Mule runtimes for the environment

C. They must be triggered by an external client tool or event source

D. They are typically written using MUnit to run in an embedded Mule runtime that does not require external connectivity

ANSWER: D

Explanation:

They are typically written using MUnit to run in an embedded Mule runtime that does not require external connectivity. MUnit is MuleSoft's testing framework for developing automated tests for Mule applications, including focused unit tests that validate application behavior in isolation. An MUnit test runs the Mule flow under test within the MUnit/Mule test runtime, so the test can invoke the flow directly rather than requiring a separately deployed application or a live external client to initiate processing.

A central unit-testing capability is mocking. MUnit can mock message processors and connector operations, returning configured payloads, attributes, variables, or errors instead of making calls to actual databases, SaaS platforms, HTTP services, queues, or other dependencies. This isolation makes unit tests deterministic, repeatable, and suitable for local execution and continuous-integration pipelines without relying on network availability or the state of external systems. MUnit also provides assertions and verification tools to confirm the output and interactions produced by the tested flow. MuleSoft documents MUnit as the framework for creating and running tests for Mule applications and documents mocking specifically as a way to simulate processor behavior during tests. See the MUnit documentation and Mocking Event Processors.

QUESTION NO: 11

An API experiences a high rate of client requests (TPS) vwth small message paytoads. How can usage limits be imposed on the API based on the type of client application?

A. Use an SLA-based rate limiting policy and assign a client application to a matching SLA tier based on its type

B. Use a spike control policy that limits the number of requests for each client application type

C. Use a cross-origin resource sharing (CORS) policy to limit resource sharing between client applications, configured by the client application type

D. Use a rate limiting policy and a client ID enforcement policy, each configured by the client application type

ANSWER: A

Explanation:

Use an SLA-based rate limiting policy and assign a client application to a matching SLA tier based on its type. In API Manager, an SLA-based rate limiting policy is designed to apply different request quotas and throughput limits to consuming applications according to the SLA tier assigned to each application. For example, internal applications, partner applications, and premium external applications can be placed in separate tiers with appropriately different requests-per-minute or requests-per-second limits. The policy identifies the requesting application through client credentials and enforces the limits associated with that application's assigned tier, allowing API owners to govern consumption consistently while supporting different client categories. This approach is particularly suitable for high-transaction APIs with small payloads because the governing requirement is request volume rather than message size. API Manager enables an organization to define SLA tiers for an API and assign client applications to those tiers, centralizing usage entitlement and enforcement rather than requiring separate API configurations for each application type. See MuleSoft's documentation for the SLA-based rate limiting and throttling policies and for managing API SLA tiers.