PassQueen
a.k.a dumpsqueen.com

MuleSoft Certified Integration Architect - Level 1 MAINTENANCE

Mulesoft MCIA-Level-1-Maintenance

Version Demo

Total Demo Questions: 16

Total Premium Questions: 166

Buy Premium PDF

https://passqueen.com

support@passqueen.com

QUESTION NO: 1

An organization is deploying a public API implementation to CloudHub. The API must be accessible through the organization-owned DNS name, and internet clients must be presented with the organization-owned server certificate.

Which two CloudHub configuration capabilities are required?

(Choose two answers)

A. Deploy an Anypoint VPC

B. Configure a CloudHub Dedicated Load Balancer with the custom domain and server certificate

C. Configure an application-level keystore only

D. Use the CloudHub Shared Load Balancer with a customer-uploaded certificate

E. Create an Anypoint MQ exchange for inbound HTTPS traffic

ANSWER: A B

Explanation:

Deploy an Anypoint VPC is correct because a CloudHub dedicated load balancer is deployed within an Anypoint VPC. The VPC provides the required private networking boundary for this load-balancing configuration.

Configure a CloudHub Dedicated Load Balancer with the custom domain and server certificate is also correct. The dedicated load balancer supports custom domains and customer-managed SSL/TLS certificates for public inbound traffic. The shared load balancer exposes the CloudHub-managed endpoint and does not provide this custom public certificate capability. MuleSoft documents these requirements in the CloudHub Dedicated Load Balancer documentation.

QUESTION NO: 2

As an enterprise architect, what are the two reasons for which you would use a canonical data model in the new integration project using Mulesoft Anypoint platform (choose two answers)

A. To have consistent data structure aligned in processes

B. To isolate areas within a bounded context

C. To incorporate industry standard data formats

D. There are multiple canonical definitions of each data type

E. Because the model isolates the backend systems and supports Mule applications from change

ANSWER: A E

Explanation:

A canonical data model provides a shared, stable representation of business data that integration flows can use consistently. “To have consistent data structure aligned in processes” is correct because a common definition for core entities and fields reduces transformation duplication and ensures that the systems participating in an end-to-end business process interpret exchanged data in the same way. This improves reuse and makes integrations easier to govern as the application landscape grows.

“Because the model isolates the backend systems and supports Mule applications from change” is also correct. MuleSoft promotes a layered, API-led approach in which consumers are shielded from underlying implementation details. By transforming backend-specific payloads into a canonical representation at the appropriate integration boundary, changes to a system’s internal schema, vendor, or interface can be contained. Consumers and downstream Mule applications can

continue to use the stable canonical contract, reducing the scope, cost, and risk of change. The canonical model should represent the enterprise's agreed business concepts and be governed as a reusable contract, rather than merely copying individual backend data structures. See MuleSoft's API-led connectivity documentation and its data transformation documentation.

QUESTION NO: 3

An organization has defined a common object model in Java to mediate the communication between different Mule applications in a consistent way. A Mule application is being built to use this common object model to process responses from a SOAP API and a REST API and then write the processed results to an order management system.

The developers want Anypoint Studio to utilize these common objects to assist in creating mappings for various transformation steps in the Mule application.

What is the most idiomatic (used for its intended purpose) and performant way to utilize these common objects to map between the inbound and outbound systems in the Mule application?

A. Use JAXB (XML) and Jackson (JSON) data bindings

B. Use the WSS module

C. Use the Java module

D. Use the Transform Message component

ANSWER: D

Explanation:

Use the Transform Message component. Transform Message is Mule's intended mechanism for implementing message transformations with DataWeave, including transformations between SOAP-derived XML, REST-derived JSON, Java values, and the data structure required by a downstream order-management system. The shared Java object model can be packaged as a dependency of the Mule application, allowing Anypoint Studio and DataWeave to use the Java types as metadata when designing mappings. This gives developers type-aware input and output structures in the mapping editor and enables DataWeave expressions to construct and populate Java objects directly when the transformation output is Java.

This approach keeps mapping logic declarative, centralized in DataWeave scripts, and optimized by the Mule runtime rather than requiring custom imperative Java mapping code throughout flows. It also supports a clear canonical-model pattern: each inbound API format is transformed into the common Java model, and the common model is then transformed into the representation expected by the outbound system. Mule documents Transform Message as the component for DataWeave transformations and documents the Java format for reading and writing Java objects and collections. See Mule Runtime Transform Message documentation and DataWeave Java format documentation.

QUESTION NO: 4

A development team is building a Mule application with MUnit tests. The tests must verify error handling for a flow that invokes an external HTTP service.

What two MUnit practices allow the team to test this error handling without depending on the external service?

(Choose two answers)

A. Mock the HTTP Request operation

B. Configure the mock to return the expected Mule error

C. Run the test only against the production endpoint

D. Disable the flow error handler during the test

E. Use a Logger component as a replacement for the HTTP Request operation

ANSWER: A B

Explanation:

Mock the HTTP Request operation is correct because MUnit can replace a connector operation with controlled behavior during a test. This prevents the test from making a network call to the external dependency.

Configure the mock to return the expected Mule error is also correct because the test can deliberately simulate the downstream failure that the application's error handler must process. The test can then assert the response payload, variables, status, or other expected handling result. This produces deterministic unit tests that do not rely on remote-system availability. MuleSoft documents connector mocking in Mocking Event Processors and assertions in the MUnit assertion documentation.

QUESTION NO: 5

An organization is designing a public Experience API that accepts requests from mobile applications. The API must prevent an individual client application from exceeding its contracted request rate and must reject clients that do not identify themselves with approved credentials.

What two API Manager policies are appropriate?

(Choose two answers)

A. Client ID enforcement

B. Rate limiting

C. JSON threat protection

D. Header injection and removal

E. CORS

ANSWER: A B

Explanation:

Client ID enforcement is appropriate because it requires API consumers to provide credentials for a registered client application. This allows the API gateway to identify the consuming application and enforce access based on the client credentials.

A rate limiting policy is appropriate because it limits the number of requests that a client can make in a configured period. Used with client identification, it can apply the contracted usage limit to each client rather than applying a single shared limit to all mobile traffic. See the Client ID-based policies documentation and Rate limiting policy documentation.

QUESTION NO: 6

An ABC Farms project team is planning to build a new API that is required to work with data from different domains across the organization.

The organization has a policy that all project teams should leverage existing investments by reusing existing APIs and related resources and documentation that other project teams have already developed and deployed.

To support reuse, where on Anypoint Platform should the project team go to discover and read existing APIs, discover related resources and documentation, and interact with mocked versions of those APIs?

A. Design Center

B. API Manager

C. Runtime Manager

D. Anypoint Exchange

ANSWER: D

Explanation:

Anypoint Exchange is correct because it is MuleSoft's central catalog for discovering and reusing organizational assets. Teams can search Exchange for published API specifications, API instances, connectors, templates, examples, documentation, and other reusable assets. An API asset's Exchange page provides its descriptive material and published documentation, allowing consumers to understand the API's purpose, available operations, data contracts, security requirements, and usage guidance before beginning implementation.

Exchange also supports API mocking for API specifications. When an asset has an API specification that can be mocked, consumers can use the mocking capability from Exchange to send requests to a simulated endpoint and inspect representative responses. This enables dependent teams to begin designing, testing, and validating integrations without waiting for a live implementation or access to underlying systems. Centralizing these discoverability, documentation, and mock-interaction capabilities in Exchange directly supports an API-led reuse strategy and organizational governance around existing investments.

MuleSoft describes Exchange as the platform's catalog for sharing and discovering reusable assets in its Anypoint Exchange documentation. The mocking workflow and its availability through Exchange are documented in the API mocking service documentation.

QUESTION NO: 7

A system API EmployeeSAPI is used to fetch employee's data from an underlying SQL database.

The architect must design a caching strategy to query the database only when there is an update to the employees stable or else return a cached response in order to minimize the number of redundant transactions being handled by the database.

What must the architect do to achieve the caching objective?

A. Use an On Table Row source on the employees table to call an invalidate-cache flow.
Use an object store caching strategy and leave the expiration interval empty.

B. Use a Scheduler with a fixed frequency every hour triggering an invalidate cache flow
Use an object store caching strategy and expiration interval to empty

C. Use a Scheduler with a fixed frequency every hour triggering an invalidate cache flow
Use an object store caching strategy and set expiration interval to 1-hour

D. Use an on table rule on employees table call invalidate cache and said new employees data to cache
Use an object store caching strategy and set expiration interval to 1-hour

ANSWER: A

Explanation:

Use an On Table Row source on the employees table to call an invalidate-cache flow, together with an object store caching strategy whose expiration interval is left empty. This design makes cache invalidation data-driven: the database listener monitors the employee table for changes and invokes the invalidation flow when a relevant row change is detected. The next request to EmployeeSAPI then misses the invalidated cache entry, retrieves current employee data from SQL, and stores that response in the cache again. Until another employee-table update occurs, subsequent requests can be served from the cached response, avoiding repeated database transactions for unchanged data.

Leaving expiration unset is important to this objective because cache freshness is governed by the detected table update rather than an arbitrary time-to-live. Mule's Cache scope supports an object-store-based caching strategy for retaining reusable responses, while cache invalidation enables an application to remove entries when the underlying data changes. The Database connector's On Table Row source is designed to poll a table for row changes and can therefore drive the invalidation flow. See MuleSoft's Cache Scope documentation and the Database Connector On Table Row documentation.

QUESTION NO: 8

An insurance company is using a CloudHub runtime plane. As a part of requirement, email alert should

be sent to internal operations team every time of policy applied to an API instance is deleted As an integration architect suggest on how this requirement be met?

A. Use audit logs in Anypoint platform to detect a policy deletion and configure the Audit logs alert feature to send an email to the operations team

B. Use Anypoint monitoring to configure an alert that sends an email to the operations team every time a policy is deleted in API manager

C. Create a custom connector to be triggered every time of policy is deleted in API manager

D. detect the policy deletion and send an email to operations team the SMTP connector

ANSWER: A

Explanation:

Use audit logs in Anypoint platform to detect a policy deletion and configure the Audit logs alert feature to send an email to the operations team is correct because deleting a policy from an API instance is an administrative action performed in Anypoint API Manager, rather than a runtime event emitted by the CloudHub application. Anypoint Platform Audit Logging records these platform-level actions, including the actor, action, resource, and timestamp. An organization administrator can use audit-log alerting to define a filter that identifies the applicable API Manager policy-deletion activity and configure email notification recipients for the internal operations team. This provides an operationally appropriate, platform-native control: the notification is generated from the authoritative record of the administrative change and does not depend on modifying the deployed API implementation. CloudHub remains relevant as the runtime plane for the APIs, but it is not the source of truth for API Manager policy administration. Audit logging and its alert configuration support governance, operational visibility, and timely response to changes affecting managed API instances. MuleSoft documents the captured administrative activity in its Audit Logging documentation and describes alert configuration in Audit Log Alerts.

QUESTION NO: 9

An API implementation is deployed to CloudHub and uses API Manager policies through autodiscovery.

What two configuration items are required to associate the deployed Mule application with its API instance?

(Choose two answers)

A. An API instance created in API Manager

B. The API instance ID configured in the Mule application's autodiscovery configuration

C. A dedicated load balancer for the application

D. A Mule domain project containing the API specification

E. An MUnit test suite deployed with the application

ANSWER: A B

Explanation:

The API must have an API instance created in API Manager, and the Mule application must be configured with the corresponding API ID in its autodiscovery configuration. The API ID identifies the managed API instance whose policies and configuration the runtime retrieves.

Autodiscovery connects the deployed Mule application to API Manager so that applicable policies can be enforced without implementing those controls manually in the application flow. The API instance and application must be in the applicable Anypoint Platform organization and environment context. See the Autodiscovery documentation.

QUESTION NO: 10

Insurance organization is planning to deploy Mule application in MuleSoft Hosted runtime plane. As a part of requirement , application should be scalable . highly available. It also has regulatory requirement which demands logs to be retained for at least 2 years. As an Integration Architect what step you will recommend in order to achieve this?

A. It is not possible to store logs for 2 years in CloudHub deployment. External log management system is required.

B. When deploying an application to CloudHub , logs retention period should be selected as 2 years

C. When deploying an application to CloudHub, worker size should be sufficient to store 2 years data

D. Logging strategy should be configured accordingly in log4j file deployed with the application.

ANSWER: A

Explanation:

It is not possible to store logs for 2 years in CloudHub deployment. External log management system is required. MuleSoft-hosted runtime deployments provide platform log viewing and retention capabilities, but these are not intended to meet multi-year regulatory archival requirements. CloudHub log storage has platform-managed limits; therefore, application logs eventually age out or are removed when storage limits are reached. Increasing worker capacity or configuring a Log4j logging strategy does not extend the platform's managed log-retention window to two years.

The integration architecture should send application and audit-relevant events to an external log-management or SIEM platform that is governed by the organization's retention, access-control, encryption, immutability, and search requirements. This can be implemented through an appropriate logging appender or log-forwarding pattern, with careful avoidance of sensitive data in logs. The external platform can then retain records for the required two-year period while CloudHub continues to provide scalable deployment and high-availability capabilities appropriate to the selected deployment configuration. MuleSoft documents CloudHub log access and retention limitations in its CloudHub 2.0 log documentation. For broader guidance on routing and managing logs from Mule applications, see MuleSoft's Mule Runtime logging documentation.

QUESTION NO: 11

A Mule application is built to support a local transaction for a series of operations on a single database. The mule application has a Scatter-Gather scope that participates in the local transaction.

What is the behavior of the Scatter-Gather when running within this local transaction?

A. Execution of all routes within Scatter-Gather occurs in parallel Any error that occurs inside Scatter-Gather will result in a roll back of all the database operations

B. Execution of all routes within Scatter-Gather occurs sequentially Any error that occurs inside Scatter-Gather will be handled by error handler and will not result in roll back

C. Execution of all routes within Scatter-Gather occurs sequentially Any error that occurs inside Scatter-Gather will result in a roll back of all the database operations

D. Execution of all routes within Scatter-Gather occurs in parallel Any error that occurs inside Scatter-Gather will be handled by error handler and will not result in roll back

ANSWER: C

Explanation:

Execution of all routes within Scatter-Gather occurs sequentially Any error that occurs inside Scatter-Gather will result in a roll back of all the database operations is correct. A local transaction coordinates operations against a single transactional resource, such as one database connection, and requires those operations to share a consistent transaction context. Although Scatter-Gather normally processes its routes concurrently, Mule changes that behavior when the scope participates in a transaction: the routes run sequentially so that the transaction can be preserved across the operations. This enables the database work performed by the routes to be treated as one atomic unit of work.

If route processing produces an unhandled error, the transaction is marked for rollback. Consequently, database changes made earlier in that transaction are not committed, preserving consistency for the complete series of operations. This behavior is particularly important where several dependent writes must either all succeed or leave the database unchanged. Mule's transaction documentation describes local transactions as transactions for a single resource and explains the rollback behavior for transaction failures. The Scatter-Gather documentation also describes its normal concurrent execution model and its behavior when used with transactional processing. See Mule transaction management and Scatter-Gather scope.

QUESTION NO: 12

A trading company handles millions of requests a day. Due to nature of its business, it requires excellent

performance and reliability within its application.

For this purpose, company uses a number of event-based API's hosted on various mule clusters that communicate across a shared message queue sitting within its network.

Which method should be used to meet the company's requirement for its system?

A. XA transactions and XA connected components

B. JMS transactions

C. JMS manual acknowledgements with a reliability pattern

D. VM queues with reliability pattern

ANSWER: C

Explanation:

JMS manual acknowledgements with a reliability pattern is appropriate for high-volume, clustered event processing that uses a shared JMS message queue. Manual acknowledgement gives the Mule application control over when a consumed message is acknowledged to the broker. The application can therefore acknowledge the message only after the required processing has completed successfully. If processing fails before acknowledgement, the broker can redeliver the message, supporting reliable at-least-once delivery across the cluster.

Combining this acknowledgement approach with a Mule reliability pattern helps manage failures in downstream processing while preserving the message until its work is safely completed. This is especially valuable for trading workloads, where a lost event can have material business consequences and where multiple nodes must consume from shared, durable infrastructure. The approach also avoids making every operation part of a distributed transaction, which can impose substantial coordination overhead at very high request volumes. Mule's JMS connector documents manual acknowledgement as an acknowledgement mode controlled by the application, and Mule reliability patterns describe the design goal of ensuring messages are retained and recoverable when failures occur.

See the MuleSoft JMS acknowledgement documentation and the Mule runtime reliability patterns documentation.

QUESTION NO: 13

An organization publishes API specifications to Anypoint Exchange and wants to ensure that API consumers can identify whether a new release is backward compatible.

Which two semantic versioning practices are correct?

(Choose two answers)

A. Increment the minor version when backward-compatible functionality is added

B. Increment the major version when incompatible API changes are introduced

C. Increment the patch version whenever a required request field is removed

D. Increment the major version for a backward-compatible documentation correction only

E. Use the same version number when the API contract changes

ANSWER: A B

Explanation:

Incrementing the minor version is appropriate when adding backward-compatible functionality. For example, adding an optional resource or an optional response field without breaking existing consumers can result in a change from 2.3.0 to 2.4.0.

Incrementing the major version is appropriate for incompatible API changes, such as removing a resource, changing a required request field, or changing a response in a way that breaks existing clients. Semantic versioning communicates compatibility expectations so API consumers can plan adoption and migration appropriately. See the Semantic Versioning specification and Anypoint Exchange documentation.

QUESTION NO: 14

An organization is defining an API versioning strategy for APIs published to Anypoint Exchange.

Which two changes normally require a new major version of an API under semantic versioning practices?

(Choose two answers)

A. Removing an existing resource

B. Adding an optional response field

C. Adding a new optional resource

D. Changing an existing response field from a string to an object

E. Correcting a spelling error in an API description

ANSWER: A D

Explanation:

Removing an existing resource and **changing an existing response field from a string to an object** are correct because both are backward-incompatible changes. Existing clients can depend on the removed resource and on the original response type, so either change can cause an already deployed client to fail.

A major version communicates that consumers might need to modify their integrations before adopting the release. Adding an optional resource or an optional response field is generally backward compatible and is normally represented by a minor-version increment. Semantic Versioning defines a major version increase for incompatible API changes; see the Semantic Versioning specification. MuleSoft documents versioned assets in Anypoint Exchange.

QUESTION NO: 15

An organization designing a hybrid, load balanced, single cluster production environment. Due to performance service level agreement goals, it is looking into running the Mule applications in an active-active multi node cluster configuration.

What should be considered when running its Mule applications in this type of environment?

A. All event sources, regardless of time , can be configured as the target source by the primary node in the cluster

B. An external load balancer is required to distribute incoming requests throughout the cluster nodes

C. A Mule application deployed to multiple nodes runs in an isolation from the other nodes in the cluster

D. Although the cluster environment is fully installed configured and running, it will not process any requests until an outage condition is detected by the primary node in the cluster.

ANSWER: B

Explanation:

An external load balancer is required to distribute incoming requests throughout the cluster nodes. In an active-active Mule cluster, application replicas are running simultaneously on multiple nodes so that the environment can increase request-handling capacity and remain available if a node becomes unavailable. The Mule cluster coordinates members and provides shared clustering capabilities, but it is not itself the network entry point that accepts a client connection and selects the appropriate runtime node. A load balancer placed in front of the cluster provides that function by routing inbound HTTP(S) traffic across healthy nodes according to its configured algorithm and health checks. This topology prevents all traffic from being directed to a single runtime and allows traffic to continue reaching available nodes when one node is removed from service. The load balancer should be configured with appropriate listener ports, health monitoring, and—where the application requires it—session affinity or other persistence considerations. MuleSoft's clustering documentation describes clusters as multiple Mule runtime instances working together for high availability and scalability, while the deployment architecture documentation identifies the external load balancer as the component used to distribute inbound traffic among cluster members. See Mule runtime clustering overview and Mule deployment architecture.

QUESTION NO: 16

As a part of project requirement, Java Invoke static connector in a mule 4 application needs to invoke a static method in a dependency jar file. What are two ways to add the dependency to be visible by the connectors class loader?

(Choose two answers)

A. In the Java Invoke static connector configuration, configure a path and name of the dependency jar file

B. Add the dependency jar file to the java classpath by setting the JVM parameters

C. Use Maven command to include the dependency jar file when packaging the application

D. Configure the dependency as a shared library in the project POM

E. Update mule-artefact.json to export the Java package

ANSWER: B D

Explanation:

Adding the dependency jar file to the Java classpath by setting JVM parameters makes the library available from the parent JVM class loader. Because Mule plugin class loaders delegate to their parent class loaders, a class placed on this JVM-level classpath can be resolved when the Java Invoke connector loads the requested static method. This approach is appropriate when the dependency is intentionally installed and managed at the runtime/JVM level.

Configuring the dependency as a shared library in the project POM is the application-scoped Mule 4 approach. Mule applications and their plugins use class-loader isolation, so a normal application dependency is not automatically visible to a connector plugin. Declaring a Maven dependency in the `sharedLibraries` section causes Mule to expose that library to plugin class loaders, allowing the connector to load the static class while retaining dependency packaging and version management with the application. MuleSoft documents shared libraries specifically for dependencies that must be accessible to an application and its plugins. See Mule Maven Plugin shared libraries and Mule class loader isolation.