DUMPSARENA

Salesforce Certified MuleSoft Developer 1

Salesforce Salesforce-MuleSoft-Developer-I

Version Demo

Total Demo Questions: 28

Total Premium Questions: 283

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Designing APIs	44
Topic 2, Building APIs	61
Topic 3, Deploying and Managing APIs	28
Topic 4, Accessing and Modifying Mule Messages	117
Topic 5, Error Handling	33
Total	283

QUESTION NO: 1

Acompany has an API to manage purchase orders, with each record identified by a unique purchase order ID. The API was built with RAML according to MuleSoft best practices.

What URI should a web client use to request order P05555?

A. /orders/{P05555}

B. /orders/order=P05555

C. /orders?order=P05555

D. /orders/P05555

ANSWER: D

Explanation:

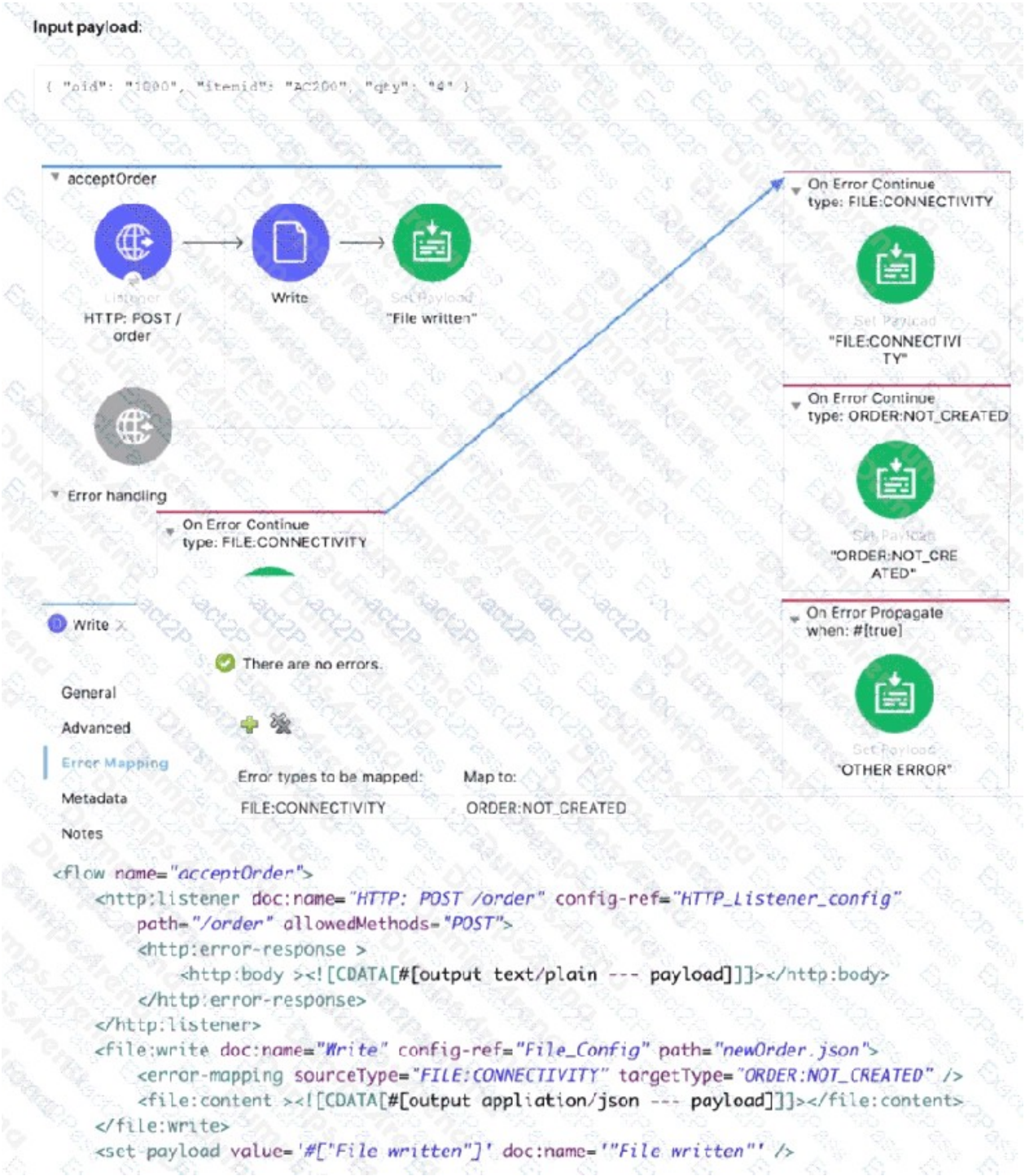
`/orders/P05555` is correct because the purchase order ID uniquely identifies one specific purchase-order resource. RESTful URI design uses a path parameter to address an individual member of a collection: the collection is represented by `/orders`, and appending the resource identifier creates the URI for one order, `/orders/{orderId}` in the RAML resource definition. When a client substitutes `P05555` for that parameter, the request URI is `/orders/P05555`. This resource-oriented structure makes the hierarchy and identity of the requested resource clear, and it maps naturally to a GET operation that retrieves a single order. RAML supports URI parameters in resource paths, with a parameter value supplied as an actual path segment by the client rather than including the curly-brace placeholder in the request. MuleSoft's API design guidance also emphasizes using nouns for resource paths and using path parameters to identify a specific resource. See MuleSoft's API design reference and the RAML 1.0 tutorial.

QUESTION NO: 2

Refer to the exhibits. The Mule application does NOT define any global error handlers.

A web client sends a POST request to the Mule application with this input payload. The File Write operation throws a FILE: CONNECTIVITY error.

What response message is returned to the web client?



A. " FILE: CONNECTIVITY "

B. " OTHER ERROR "

C. " File written "

D. " ORDER: NOT CREATED "

ANSWER: D

Explanation:

"ORDER: NOT CREATED" is returned because the error raised by the File Write operation is handled through the application's nested error-handling scopes. The File Write operation raises an error of type `FILE:CONNECTIVITY`. The error handler associated with the Try scope handles that condition and uses an `On-Error Propagate` scope, so the error continues beyond the Try scope rather than allowing normal processing to resume there. The propagated error retains its

error type as it reaches the enclosing flow's error handler. The flow-level handler has a matching handler for `FILE:CONNECTIVITY`, which sets the payload to `ORDER: NOT CREATED`. Because that enclosing handler handles the propagated error and completes the flow's error handling, its payload becomes the HTTP response body sent to the web client. This behavior does not require a global error handler: Mule resolves errors first in the nearest applicable scope and then propagates them to enclosing scopes when propagation is configured. MuleSoft documents the distinction between continuing and propagating errors, as well as how matching error types select an error handler, in its On-Error Scope documentation and error handling overview.

QUESTION NO: 3

Refer to the exhibits.

The web client sends a POST request to the ACME Order API with an XML payload. An error is returned.

What should be changed in the request so that a success response code is returned to the web client?

A. Set a request header with the name Content-Type to a value of applicatron/octet-stream

B. Set a request header with the name Content-Type to a value of application/xml

C. Set a response header with the name Content-Type to a value of applkation/xml

D. Set a response header with the name Content-Type to a value of application/octet-stream

ANSWER: B

Explanation:

Set a request header with the name Content-Type to a value of application/xml. The `Content-Type` request header tells the API how to interpret the representation in the HTTP request body. Because the client is sending XML, it must declare the XML media type as `application/xml`. In a RAML-defined Mule API, the request body media type is part of the API contract; when the API expects XML, declaring `application/xml` allows Mule's HTTP listener and API implementation to select the appropriate body handling and validate or process the payload as XML. A missing or incompatible media type can result in an HTTP 415 Unsupported Media Type response before the request reaches the intended processing flow. The header belongs on the request sent by the web client, because it describes the payload being submitted. Once the XML body and its declared media type agree, the API can accept and process the POST request, allowing it to return its configured success response. See the HTTP semantics definition of `Content-Type` at RFC 9110 and MuleSoft guidance on defining request body media types in RAML at MuleSoft Documentation.

QUESTION NO: 4

Refer to the exhibits.

The Batch Job scope processes the array of strings

After the Batch Job scope completes processing the input payload what information is logged by the Logger component?

A)

B)

C)

D)

A. Option A

B. Option B

C. Option C

D. Option D

E. The required logger output cannot be determined because the exhibits are missing.

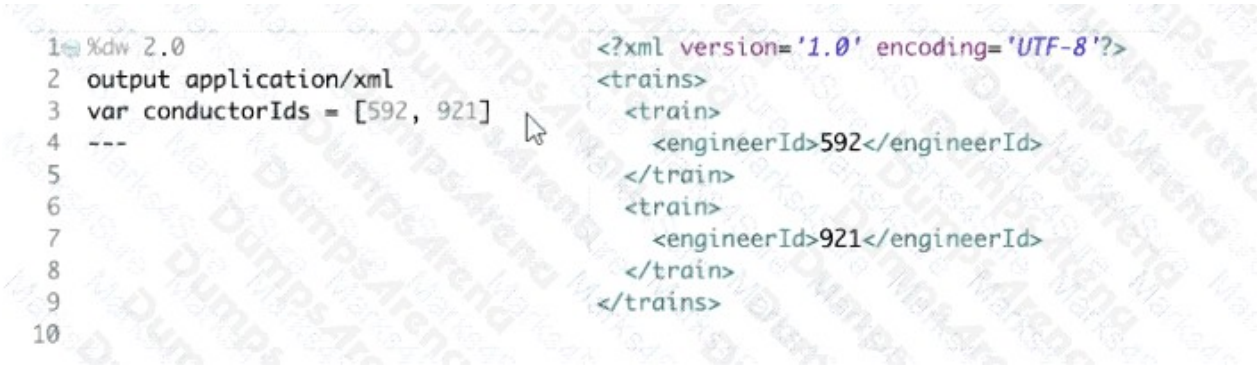
ANSWER: E

Explanation:

The required logger output cannot be determined because the exhibits are missing. A Batch Job scope has distinct phases, including loading and dispatching records, processing records, and completing the job. The value visible to a Logger after that scope depends on the omitted flow configuration: specifically, the Logger message expression, any transformations before or within the batch job, whether variables are created or modified, and the batch aggregation or completion behavior shown in the missing exhibits. The question states only that an array of strings is processed, which is not enough to establish the payload, attributes, variables, record status, or any textual message that the Logger evaluates after processing completes. Mule applications can log payload content, variable values, batch-job result information, or a literal message, and each possibility produces a different result. Therefore, a definite output can only be selected after the missing XML configuration or screenshots are supplied. MuleSoft's batch-processing documentation describes the separate processing phases and completion behavior, while the Logger documentation confirms that its output is determined by the configured message expression. See MuleSoft Batch Processing and Logger Component Reference.

QUESTION NO: 5

Refer to the exhibit.



```
%dw 2.0
output application/xml
var conductorIds = [592, 921]
---
```

```
<?xml version='1.0' encoding='UTF-8'?>
<trains>
  <train>
    <engineerId>592</engineerId>
  </train>
  <train>
    <engineerId>921</engineerId>
  </train>
</trains>
```

What DataWeave expression transforms the conductorIds arrayto the XML output?

A)

```
{(
    trains: conductorIds map ( (engId, index) ->
        train: {
            engineerId: engId
        }
    )
)}
```

B)

```
trains:
    conductorIds map ( (engId, index) ->
        train: {
            engineerId: engId
        }
    )
```

C)

```
trains:
{(
    conductorIds map ( (engId, index) ->
        train: {
            engineerId: engId
        }
    )
)}
```

D)

```
{
    trains: conductorIds map ( (engId, index) ->
        train: {
            engineerId: engId
        }
    )
}
```

A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: C

Explanation:

Option C is the correct transformation. The required DataWeave expression iterates over the values in the `conductorIds` array and produces an XML element for each array item. In DataWeave, an array-to-XML transformation is typically expressed by mapping the source array to an object whose key is the desired XML element name. Each object created by the mapping operation becomes a repeated XML child element in the output. The mapped value is then used as the content or attribute value required by the target XML structure.

This approach is important because DataWeave treats repeated object keys in an XML result as repeated elements, allowing an array to be represented naturally without manually concatenating XML strings. The expression also preserves the input array order, so the generated XML nodes correspond to the sequence of conductor identifiers supplied by the payload. DataWeave's XML writer serializes the resulting object structure into well-formed XML when the output MIME type is `application/xml`.

For details on mapping arrays and constructing output objects, see the DataWeave map documentation and the DataWeave XML format documentation.

QUESTION NO: 6

Refer to the exhibits.

The Batch Job scope contains two BatchStep scopes with different accept expressions.

The input payload is passed to the Batch Job scope.

After the entire payload is processed by the Batch Job scope, what messages have been logged by the Logger components?

A)

B)

C)

D)

A. Option A

B. Option B

C. Option C

D. Option D

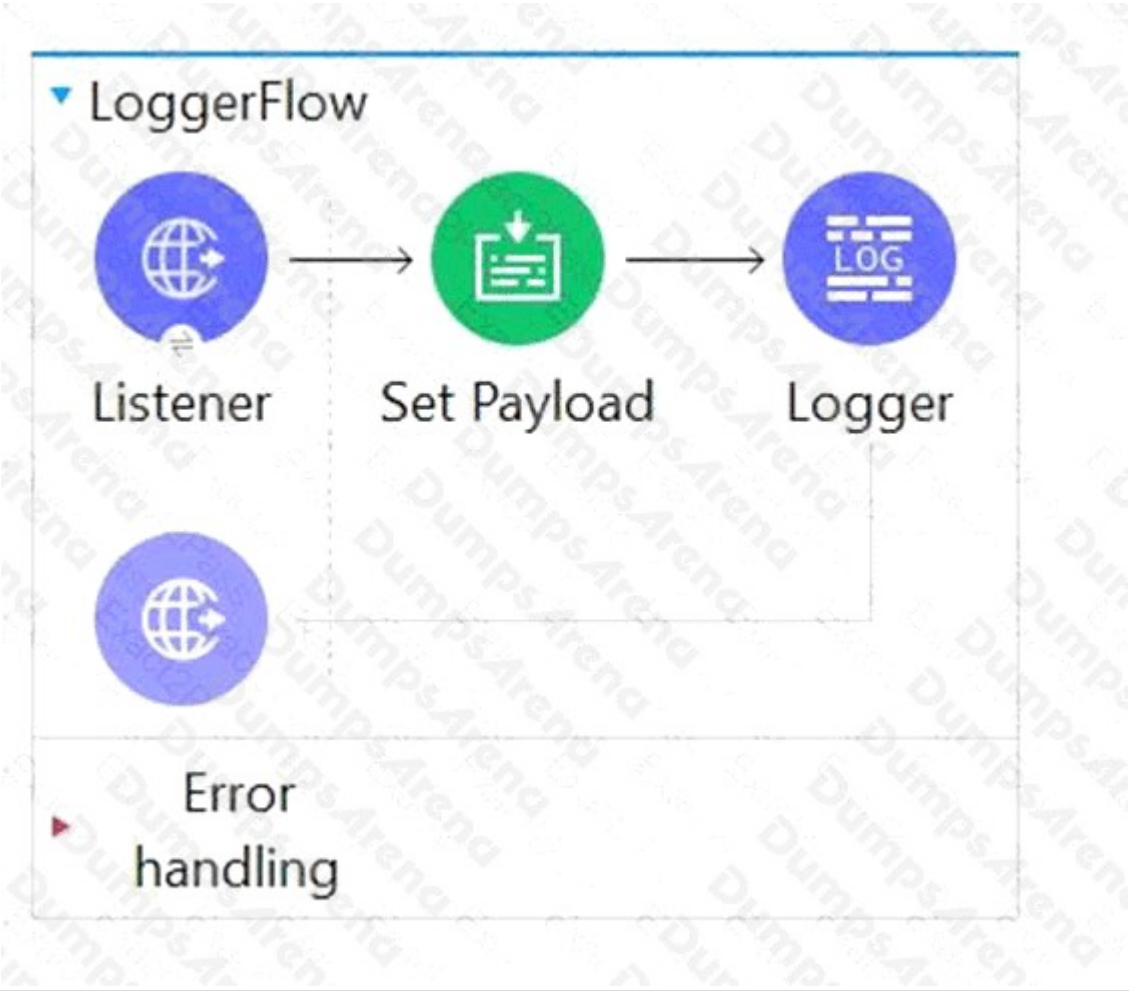
ANSWER: C

Explanation:

Option C is correct because a Mule batch job first splits the input collection into individual records, then evaluates each record independently as it enters each BatchStep. An `acceptExpression` determines whether that particular record is accepted for processing by a given step. Records that satisfy the expression execute the processors in that BatchStep, including its Logger; records that do not satisfy it simply do not enter that step. Since the two steps use different acceptance criteria, a single input record can be accepted by one step, both steps, or neither step, depending on the values shown in the exhibits. The resulting logger output is therefore the combined set of messages produced only for records accepted by each respective step, in accordance with the expressions and record data shown. Batch processing completes only after all records have been evaluated and processed through the applicable steps, so the selected result accurately represents the full set of logger messages after the batch job finishes. See MuleSoft's Batch Processing concept documentation and the Batch filters documentation for the record-level filtering behavior used by batch steps.

QUESTION NO: 7

Refer to the payload.



The Set payload transformer sets the payload to an object. The logger component ' smessage attribute is configured with the string " Result #[" INFO " ++ payload] "

What is the output of logger when this flow executes?

A. Result INFOpayload

B. Result INFO{ " student " :{ " name " : " Anay " , " age " :6}}

C. 1. 1. " You called the function ' ++ ' with these arguments:2. 2. 1: String (" INFO ")3. 3: Object ({student: {name: " Anay " as String {class: " java.lang.String " },age: 6 as Numbe...)

D. Error : You evaluated inline expression # without ++

ANSWER: C

Explanation:

The output beginning with “You called the function '++' with these arguments” is correct because the expression inside the Logger message is evaluated as DataWeave before Mule writes the log entry. The payload established by Set Payload is an Object containing the `student` structure, while `"INFO"` is a String. In DataWeave, `++` is an overloaded concatenation operator, but its operands must match a supported concatenation signature. DataWeave cannot concatenate a String directly with an Object using this expression, so evaluation fails and Mule reports the attempted function invocation, including the runtime types of the supplied values: String and Object. Consequently, the logger does not produce a normal message containing a serialized payload; instead, it reports the DataWeave expression error. This behavior is useful because the error identifies both the operator and the actual value types involved, helping a developer determine that an explicit conversion or a different expression is needed when incorporating an object into a log message. MuleSoft documents DataWeave operator behavior in its DataWeave Operators reference, and describes message logging and expression evaluation in the Logger Component reference.

QUESTION NO: 8

How many Mule applications can run on a CloudHub worker?

A. At most one

B. At least one

C. Depends

D. None of these

ANSWER: A

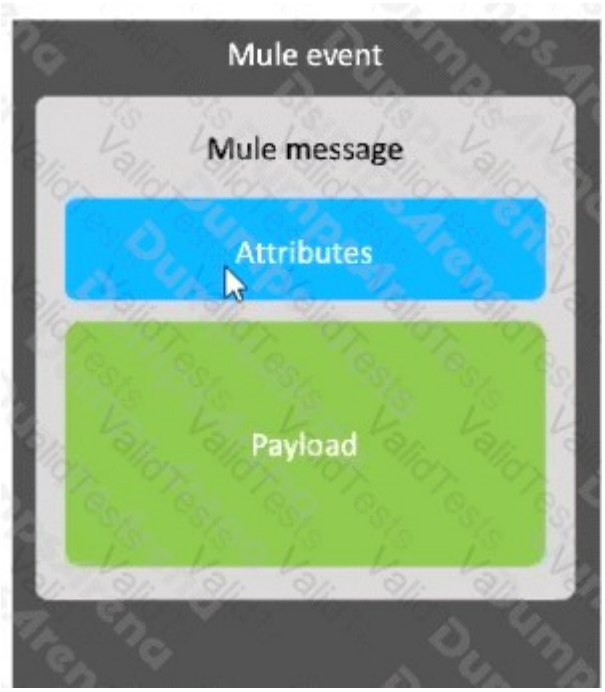
Explanation:

At most one is correct. In CloudHub, a worker is a dedicated runtime instance allocated to a deployed Mule application. CloudHub scales an application by assigning it additional workers, but each worker runs only that single application; it is not a shared container in which several separately deployed Mule applications run together. For example, if an application is configured with two workers for high availability or increased processing capacity, CloudHub runs two instances of that same application, with one application instance per worker. This isolation helps ensure that the application has dedicated runtime resources and that scaling, monitoring, logging, and lifecycle operations are managed at the application level. The number and size of workers can be selected according to the application's performance and availability requirements, but this does not change the one-application-per-worker model. MuleSoft’s CloudHub architecture documentation describes workers as instances of Mule runtime that run a deployed application, while the deployment model documentation explains worker-

based scaling for an individual application. See CloudHub architecture documentation and CloudHub deployment documentation.

QUESTION NO: 9

Refer to the exhibit.



A Mule event is composed of a hierarchy of objects. Where in the hierarchy are variables stored?

A. Mule event

B. Mule message payload

C. Mule message

D. Mule message attributes

ANSWER: A

Explanation:

Mule event is correct because a Mule event is the complete unit of data that Mule runtime processes as it moves through an application flow. The event contains the Mule message and also maintains event-scoped state, including variables. A Mule message itself is composed of a payload and attributes: the payload holds the primary business data, while attributes hold metadata supplied by the source connector, such as HTTP headers, query parameters, file information, or listener details. Variables are not part of either the payload or attributes. Instead, they belong directly to the event and can be set or updated by components such as Set Variable, then used by later processors in the same flow. This placement lets an application retain temporary contextual data separately from the message content and source metadata without changing either one. MuleSoft documents the event as containing a message, variables, and other contextual information, and its variable documentation describes variables as values stored in the Mule event for use while processing that event. See the Mule Event Structure documentation and the Mule Variables documentation.

QUESTION NO: 10

What is the object type returned by the File List operation?

A. Object of String filenames

B. Array of String file names

C. Object of Mule event objects

D. Array of Mule event objects

E. Array of File objects

ANSWER: E

Explanation:

The File List operation returns an **Array of File objects**. This operation enumerates the entries in the configured directory and places the resulting collection in the Mule event payload. Each item represents a file-system entry as a file object, rather than only exposing its filename as text. This representation enables a flow to use the returned entries in subsequent processing, such as iterating over them, selecting entries by properties, or passing an entry to another File connector operation. The operation is intended to discover files and folders; it does not read each file's content into the payload as part of the listing itself. Mule events provide the runtime message envelope that carries the payload and attributes through a flow, but the items returned by this operation are file objects, not nested Mule event objects. The File connector documentation describes List as the operation used to list the files and folders in a directory, and the connector reference identifies its output as an array. See the File Connector List operation documentation and the File Connector reference.

QUESTION NO: 11

Refer to the exhibits.

The Validation component in theTry scope throws an error.

What response message is returned to a client request to the main flow's HTTP Listener?

The Validation component in the Try scope throws an error. What response message is returned to a client request to the main flow's HTTP Listener?

A. Success - main flow

B. Error - main flow

C. Error - Try scope

D. Validation Error

ANSWER: A

Explanation:

Success - main flow is returned because the validation failure is handled by an `On Error Continue` error handler in the referenced private flow. When a processor raises an error, normal execution of that flow stops and Mule enters the matching error handler. An `On Error Continue` handles the error and considers the flow's execution successful from the perspective of the caller. It can set a payload, but it does not propagate the error back to the main flow.

Consequently, once the private flow completes its error handling, the flow reference returns control to the main flow without an active error. The main flow continues with its next processor, which sets the payload to **Success - main flow**. Since the HTTP Listener uses the final Mule message payload as its response body when no explicit response configuration overrides it, that final payload is sent to the client. The request therefore completes as a successful response rather than exposing the validation error to the HTTP caller. MuleSoft documents that `On Error Continue` executes its processors and treats the owner flow as successfully completed, enabling processing to resume in the calling flow. See On-Error Scope and HTTP Listener Configuration Reference.

QUESTION NO: 12

What is the trait name you would usefor specifying client credentials in RAML?

A. headers

B. client-id

C. client-id-required

D. cannot be specified in RAML

ANSWER: C

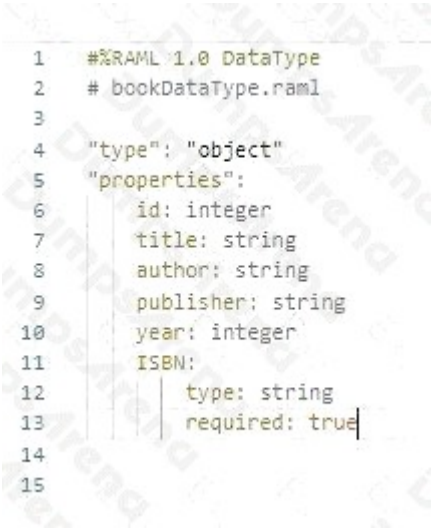
Explanation:

`client-id-required` is the RAML trait used by MuleSoft to declare that an API operation requires client application credentials. Applying this trait indicates that a consuming application must provide its client ID and client secret to access the protected resource or method. The credentials can be supplied in headers or query parameters, according to the API's client-ID enforcement configuration and the gateway's supported credential locations. In an API specification, the trait is typically imported from MuleSoft's Exchange-hosted client-ID enforcement library and then applied at the API, resource, or method level as appropriate. This makes the credential requirement explicit in the API contract, improves generated documentation, and supports client application registration and enforcement through API Manager. The trait name is specifically `client-id-required`; it represents the reusable RAML declaration for this authentication requirement. MuleSoft documents this approach in its guidance on applying client ID enforcement and defining client credentials in API specifications. See MuleSoft Client ID Enforcement policies and Anypoint Exchange documentation.

QUESTION NO: 13

A company has defined two RAML fragments, Book Data Type and Book Example to be used in APIs.

What would be valid RAML to use these fragments ?



```
#%RAML 1.0 DataType
# bookDataType.raml

"type": "object"
"properties":
    id: integer
    title: string
    author: string
    publisher: string
    year: integer
    ISBN:
        type: string
        required: true
```

```
#%RAML 1.0 NamedExample
# bookExample.raml

bookExample:
    ID: 101
    title: Shakespeare
    author: Encylopedia Britanica
    publisher: John Wiley & Sons
    year: 2007
    ISBN: "0471767840"
```

A. 1. #%RAML 1.02. title: Books3.types:4. Book: ABC/Examples/bookDataType.raml5. /books:6. post:7. body:8. application/json:9. type: Book10. examples:11. input: ABC/Examples/bookExample.raml12. responses:13. 201:14. body:15. application/json:16.example:17.message: Book added

B. 1.#%RAML 1.02.title: Books3.Book: !include bookDataType.raml4./books:5.post:6.body:7.application/json:8.type: Book9.examples:10.input: !include bookExample.raml11.responses:12.201:13.body:14.application/json:15.example:16.message: Book added

C. 1.#%RAML 1.02.title: Books3.Book: bookDataType.raml4./books:5.post:6.body:7.application/json:8.type: Book9.examples:10.input: bookExample.raml11.responses:12.201:13.body:14.application/json:15.example:16.message: Book added

D. 1.#%RAML 1.02.title: Books3.Book: bookDataType.raml4./books:5.post:6.body:7.application/json:8.type: Book9.examples:10.input: bookExample.raml11.responses:12.201:13.body:14.application/json:15.example:16.message: Book added

E. #%RAML 1.0 title: Books types: Book: !include bookDataType.raml /books: post: body: application/json: type: Book examples: input: !include bookExample.raml responses: 201: body: application/json: example: message: Book added

ANSWER: E

Explanation:

The valid approach is to use the RAML `!include` tag to incorporate each external fragment at the location where its fragment type is valid. The Book Data Type fragment is included beneath the root-level `types` declaration and assigned the reusable type name `Book`. That name can then be referenced by the request body using `type: Book`. This makes the data type available to API resources while retaining its definition in a separate, reusable RAML fragment.

The Book Example fragment is a named example, so it belongs beneath the media type's `examples` map. Assigning it the name `input` and using `!include` supplies a valid request example for the `application/json` body. RAML's include mechanism is specifically intended to modularize API specifications by inserting the content of another file, and the inserted content must be compatible with the enclosing RAML node. This structure therefore correctly uses both the data type and named-example fragments in their respective RAML contexts.

See the RAML documentation on reusing RAML fragments and the RAML API specification structure.

QUESTION NO: 14

What is the correct syntax for a Logger component to output a message with the contents of a 3SON Object payload?

A. The payload is: $(payload)

B. #["The payload is: " ++ payload]

C. The payload is: #[payload]

D. #["The payload is: " + payload]

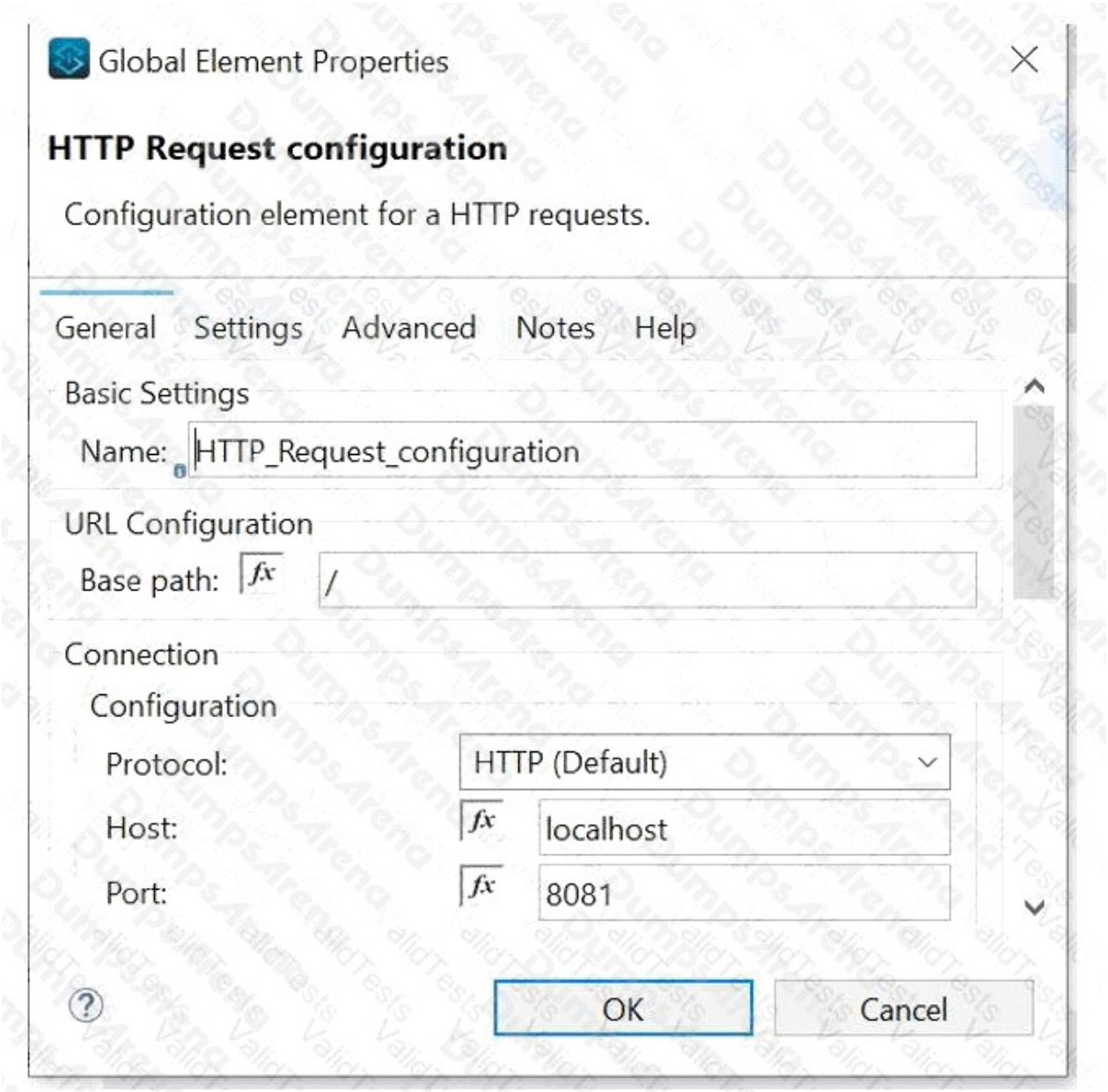
ANSWER: C

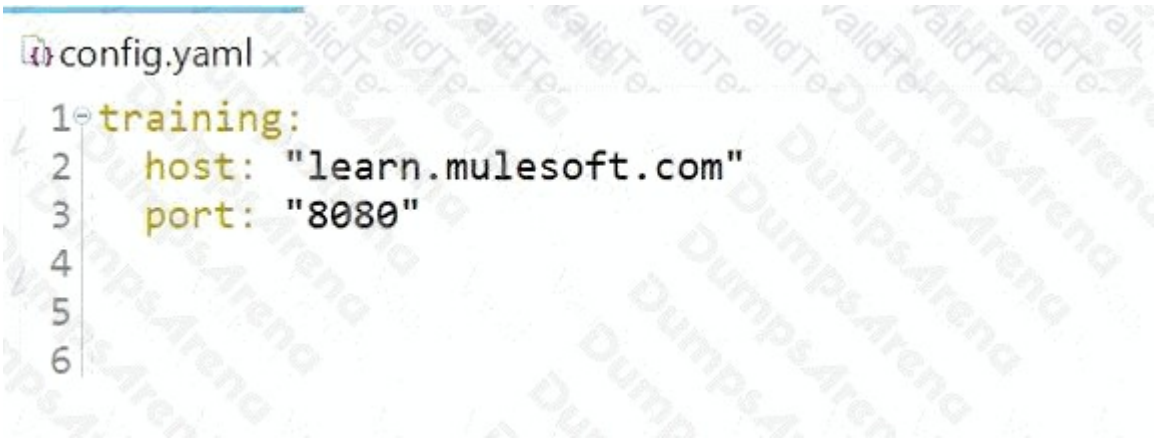
Explanation:

`The payload is: #[payload]` is the correct Logger message syntax. In Mule applications, the Logger component's Message field supports embedded DataWeave expressions using the `#[...]` delimiter. Static text can be written directly in the field, and the embedded expression is evaluated at runtime. Here, `payload` references the current Mule event payload, so the logger outputs the literal prefix followed by the payload's contents. When the payload is a JSON object, Mule uses the payload's associated representation to render it for the log message. This pattern is useful for development and troubleshooting because it lets a flow log a readable payload value without changing the event or creating a separate transformation just for logging. MuleSoft documents that Logger messages can contain DataWeave expressions, including expressions that access the current payload. See the Logger Component Reference and the DataWeave documentation.

QUESTION NO: 15

Refer to the exhibits.





Mule application has an HTTP request configurationwhere host name is hardcoded. Organization is looking to move host and port values to configuration file. What valid expression can be used to so that HTTP configuration can pick the value from configuration file?

A. #[training.host]

B. ${http.host}

C. #{training.host}

D. ${training.host}

ANSWER: D

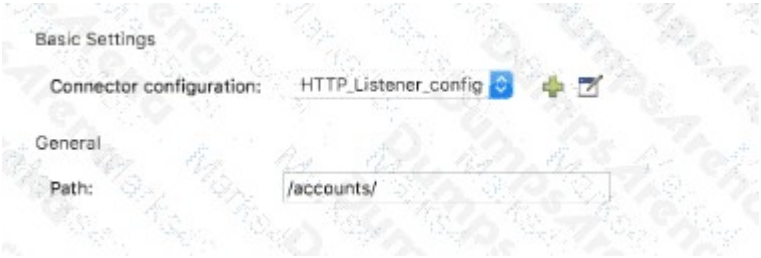
Explanation:

`${training.host}` is the valid property-placeholder expression for supplying the HTTP Request configuration's host value from an external configuration property. Mule applications commonly define environment-specific settings, such as host names and ports, in a properties file, for example `training.host=api.example.com`. A Configuration Properties element loads that file into the application's property resolution context. At application startup, Mule resolves the `${...}` placeholder and injects the resulting string into the HTTP connector configuration. This keeps endpoint details out of the flow and connector XML, allowing the same deployable application artifact to be configured differently across development, test, and production environments simply by changing property values or supplying environment-specific property sources.

The property name inside the placeholder must match the configured key exactly. Thus, when the configuration file defines the key as `training.host`, the HTTP Request host field uses `${training.host}`. The same pattern can be applied to a port key, such as `${training.port}`, in the port field. MuleSoft documents configuration properties as values loaded from property files and referenced using the `${key}` syntax. See Configuring Properties and HTTP Request Connector Reference.

QUESTION NO: 16

Refer to the exhibit.



What is the correct syntax to add an employee ID as a URI parameter in an HTTP Listener path?

A. (employeeID)

B. ${emp!oyeeID}

C. {employeeID}

D. # [employeeID]

ANSWER: C

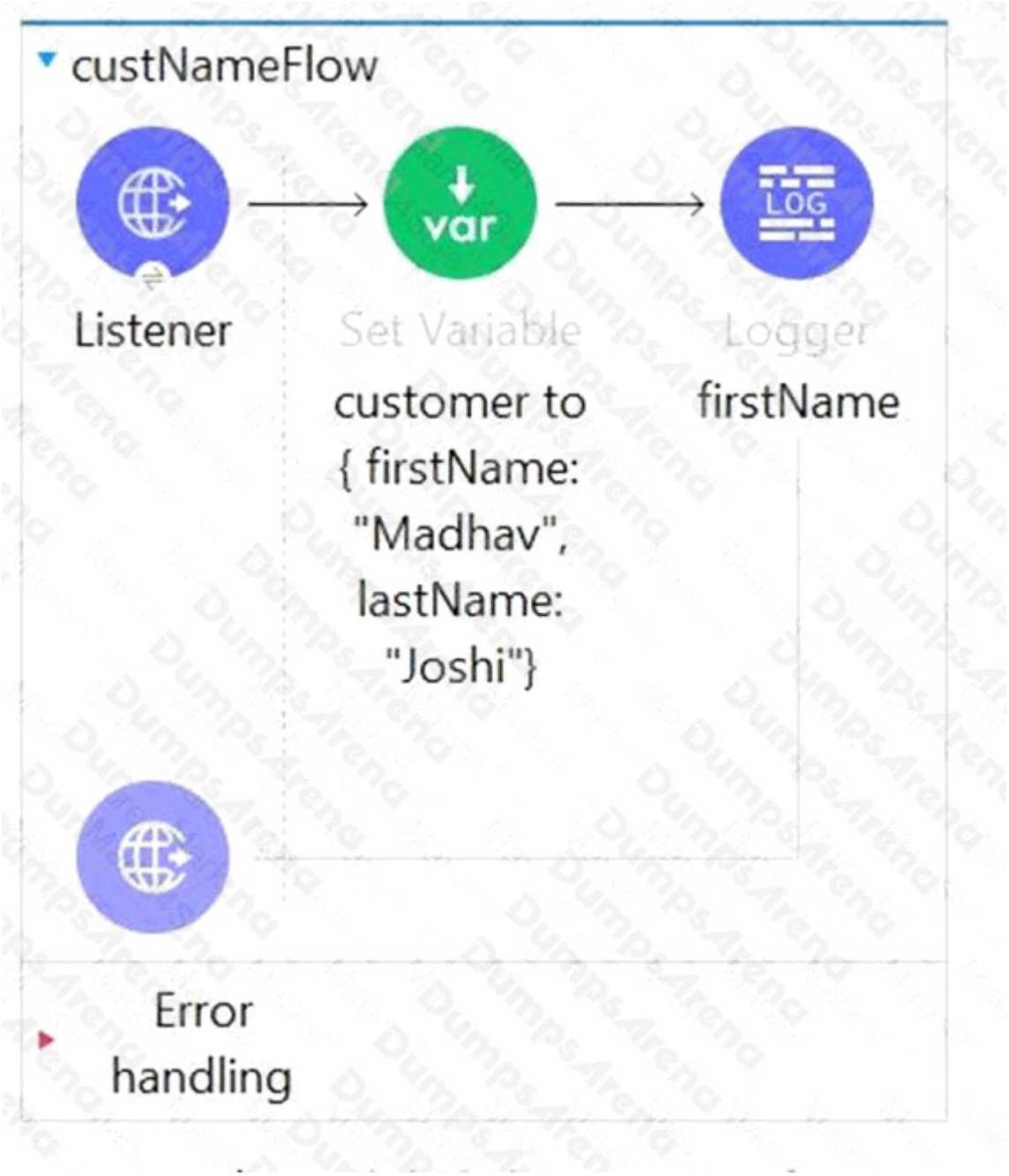
Explanation:

`{employeeId}` is the correct HTTP Listener path syntax for defining an employee ID as a URI parameter. Mule HTTP Listener paths support parameter placeholders enclosed in curly braces. When an inbound request matches that segment of the configured path, Mule captures the actual segment value and makes it available in the event's URI parameters map. For example, if a listener path is configured as `/employees/{employeeId}` and the application receives a request for `/employees/12345`, the value `12345` is captured under the key `employeeId`. It can then be accessed in a Mule expression, such as `attributes.uriParams.employeeId`, for use in routing, validation, transformations, or downstream service calls. The parameter name is developer-defined, so `employeeId` is a clear and conventional name; the essential requirement is the curly-brace placeholder format. This mechanism lets one listener serve requests for many employee

records without requiring a separate static listener path for each ID. MuleSoft documents listener path parameters as `{param}` placeholders that match a URI path segment and store its value in URI parameters. See the HTTP Listener path documentation and the Mule message structure documentation.

QUESTION NO: 17

Refer to the exhibits.




```
<flow name="custNameFlow" doc:id="bcbd3cba-7ff3-469a-a38f-826be38282ba" >
    <http:listener doc:name="Listener" config-ref="HTTP_Listener_config" path="/log"/>
    <set-variable value='#[{ firstName:"Madhav", lastName:"Joshi"}]'
    doc:name='customer to { firstName: "Madhav", lastName: "Joshi"}' variableName="customer"/>
    <logger level="INFO" doc:name="firstName"  message="??????"/>
</flow>
```

Set paylaod transformer is set the firstName and lastName of the customer as shown in below images.

What is thecorrect Dataweave expression which can be added in message attribute of a Logger activity to access firstName (which in this case is Madhav) from the incoming event?

A. firstName

B. customer.firstName

C. vars. " customer.firstName "

D. vars. " customer " . " firstName "

ANSWER: D

Explanation:

`vars. " customer " . " firstName "` is correct because the customer information is stored in a Mule event variable named `customer`. Mule applications access event variables through the `vars` object in DataWeave. The expression first selects the variable using `vars." customer "`, then selects its `firstName` field with `." firstName "`. The resulting value is the string `Madhav`, which the Logger can evaluate and write as part of its message.

DataWeave supports quoted selectors for object keys. This form is especially useful when a key contains characters that would make ordinary dot notation unsuitable, and it remains valid for conventional field names such as `customer` and `firstName`. In this case, the quoted-selector syntax traverses the variable object directly and retrieves the requested property from the incoming Mule event. MuleSoft documents that variables are accessed in DataWeave through `vars`, while object fields can be selected with dot notation or quoted field selectors. See Mule variables documentation and DataWeave selectors documentation.

QUESTION NO: 18

A Mule application calls two backup services using a First Successful router. The first route returns an HTTP 500 error, and the second route returns a successful response.

What happens after the First Successful router completes?

A. The router returns the HTTP 500 error immediately and does not try the second route

B. The router executes both routes in parallel and returns an array of results

C. The router continues with the successful response from the second route

D. The router retries the first route until it returns a successful response

ANSWER: C

Explanation:

The router continues to the second route after the first route fails, and then continues the flow using the successful result from the second route. A First Successful router attempts its routes in order until one route completes successfully.

Unlike Scatter-Gather, First Successful does not execute all routes in parallel or aggregate every route result. It also does not stop permanently after the first failure when another configured route can still be attempted.

QUESTION NO: 19

What is the minimum Cloudhub worker size that can be specified while deploying muleapplication?

A. 0.2 vCores

B. 0.5 vCores

C. 1.0 vCores

D. 0.1 vCores

ANSWER: D

Explanation:

0.1 vCores is the minimum CloudHub worker size available for an application deployment. A CloudHub worker is a dedicated Mule runtime engine instance that provides the compute and memory resources used to execute a deployed Mule application. Worker capacity is expressed in vCores, and selecting the worker size is part of configuring an application's deployment resources. The 0.1-vCore size is intended for very small workloads, such as lightweight development, testing, demonstrations, or integrations with low processing demand. It provides the lowest available allocation of compute capacity, enabling teams to deploy small applications while minimizing the vCore capacity consumed. As application traffic, flow complexity, connector use, payload sizes, or concurrency requirements increase, a larger worker size may be necessary to provide suitable memory and processing headroom. MuleSoft documents the available CloudHub worker sizes and describes workers as the runtime instances that run integration applications. See the CloudHub workers architecture documentation and the CloudHub deployment documentation for deployment configuration details.

QUESTION NO: 20

What valid RAML retrieves details on a specific customer by its customerId as a URI parameter?

A. 1. /customers:2. /get:3. /customerId:

B. 1. /customers:2. /{customerId}:3. get:

C. 1. /customers:2. /customerId:3. get:

D. 1. /customers:2. get:3. /{customerId}:

ANSWER: B

Explanation:

The valid RAML structure is `/customers:`, followed by the nested resource `/{customerId}:`, followed by `get:`. This defines a collection resource at `/customers` and then a URI-template child resource that represents one particular customer. At runtime, a request such as `GET /customers/12345` binds `12345` to the `customerId` URI parameter, allowing the implementation to identify and return that customer's details.

RAML models resource paths hierarchically through indentation. HTTP methods such as `get` are declared beneath the resource path to which they apply. Curly braces are required around a URI parameter name because they indicate a URI-template expression rather than a literal path segment. This resource arrangement is the conventional RESTful representation for reading an individual member of a customer collection. The parameter can additionally be documented with a `uriParameters` section beneath the parameterized resource when constraints, examples, or a data type are needed.

See MuleSoft's guidance on defining RAML resources and methods in the Design Center RAML API documentation and the URI-template resource syntax in the RAML 1.0 specification.

QUESTION NO: 21

A Database On Table Row listener retrieves data from a CUSTOMER table that contains a primary key userjd column and an increasing kxjin_date_time column. Neither column allows duplicate values.

How should the listener be configured so it retrieves each row at most one time?

A. Set the watermark column to the kxjin_date_time column

B. Set the target value to the last retrieved login_date_time value

C. Set the target value to thelast retrieved user_jd value

D. Set the watermark column to the user_Id column

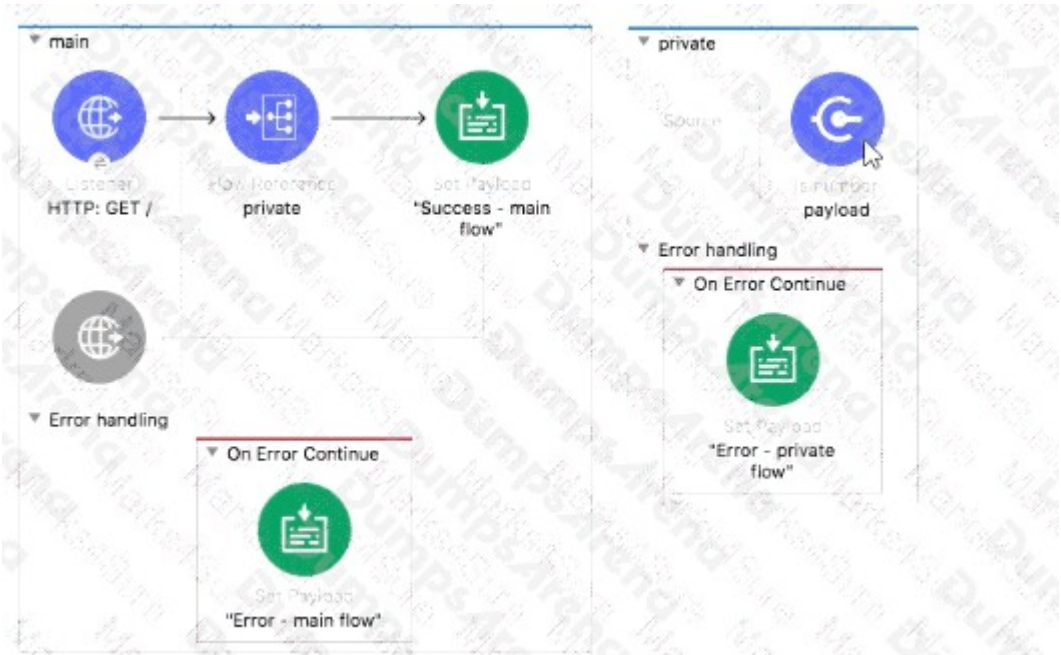
ANSWER: A

Explanation:

Set the watermark column to the kxjin_date_time column. A Database On Table Row listener uses a watermark to track the highest value successfully processed during earlier polling cycles. On subsequent polls, the listener uses that saved watermark value to select only rows whose watermark-column value is greater than the stored value. Because `kxjin_date_time` is explicitly increasing and does not allow duplicate values, it provides an ordered, unambiguous progression through the CUSTOMER rows. Each newly inserted row can therefore be identified by a value greater than the previously stored watermark, allowing the listener to advance without selecting a previously retrieved row again. Mule persists watermark state through its object store mechanism, so the listener can retain its position across polling cycles and application restarts, subject to the configured object-store persistence and deployment environment. This approach is the intended polling pattern when records must be consumed incrementally from a database table. See MuleSoft's Database On Table Row listener documentation and its watermark documentation for the listener's watermark behavior and configuration.

QUESTION NO: 22

Refer to the exhibits.




```
<flow name="main" >
    <http:listener doc:name="HTTP: GET /" config-ref="HTTP_Listener_config" path="/" />
    <flow-ref doc:name="private" name="private"/>
    <set-payload value="Success - main flow" doc:name='"Success - main flow"' />
    <error-handler>
        <on-error-continue enableNotifications="true" logException="true" doc:name="On Error Continue" >
            <set-payload value="Error - main flow" doc:name='"Error - main flow"' />
        </on-error-continue>
    </error-handler>
</flow>

<flow name="private" >
    <validation:is-number numberType="INTEGER" doc:name="payload" value="#[payload]"
    message="Validation Error" />
    <error-handler >
        <on-error-continue enableNotifications="true" logException="true" doc:name="On Error Continue" >
            <set-payload value="Error - private flow" doc:name='"Error - private flow"' />
        </on-error-continue>
    </error-handler>
</flow>
```

The Validation component in the private flow throws an error. What response message is returnedto a client request to the main flow ' s HTTP Listener?

A. Error - private flow

B. Error - main flow

C. Success - main flow

D. Validation Error

ANSWER: B

Explanation:

Error - main flow is returned because the validation failure is first handled in the private flow's processor-level error handler. Its `On Error Propagate` scope sets the payload to the private-flow error message, but propagation means the error is rethrown to the calling flow rather than being treated as successfully handled at that level. The main flow therefore receives the propagated error and transfers control to its own flow-level error handler. The main flow's `On Error Continue` scope sets the payload to `Error - main flow`. An `On Error Continue` handler treats the error as handled and allows the flow to complete normally from the perspective of the inbound HTTP request. Consequently, the HTTP Listener sends the payload established by that handler back to the client, which is `Error - main flow`. Mule's error-handling model distinguishes these two behaviors specifically: `On Error Propagate` rethrows the error to the owner or caller, whereas `On Error Continue` marks it as handled and continues normal completion. See MuleSoft's documentation on On-Error scopes and Try scope error handling.

QUESTION NO: 23

Which of the below is not a valid category for connector type?

A. Gold

B. Select

C. Premium

D. Community

ANSWER: A

Explanation:

Gold is the correct answer because it is not one of MuleSoft's connector support categories. MuleSoft classifies connectors according to their source, support model, and validation level so that developers can understand the expected ownership and support experience when selecting an integration component. The recognized categories in the referenced connector model include Select, Premium, and Community. These labels help identify whether a connector is supplied and supported by MuleSoft, offered under a premium support arrangement, or made available through the community ecosystem. "Gold" may sound like a service-tier designation, but it is not a connector classification used by MuleSoft in this model. Therefore, when evaluating connector type categories, Gold is the item that does not belong to the established set. MuleSoft's historical connector documentation describes these support categories and their intended purpose in helping teams assess connector availability and support: MuleSoft connector support categories. For current connector discovery and metadata in Anypoint Exchange, see Anypoint Exchange documentation.

QUESTION NO: 24

A Mule project contains a DataWeave module called MyModule.dwl that defines a function named formatString. The module is located in the project's src/main/resources/modules folder.

What is the correct way in DataWeave code to import MyModule using a wildcard and then call the module's formatString function?

A)

B)

C)

D)

A. Option A

B. Option B

C. Option C

D. Option D

E.
```
import * from modules::MyModule
---
formatString("example")
```

ANSWER: E

Explanation:

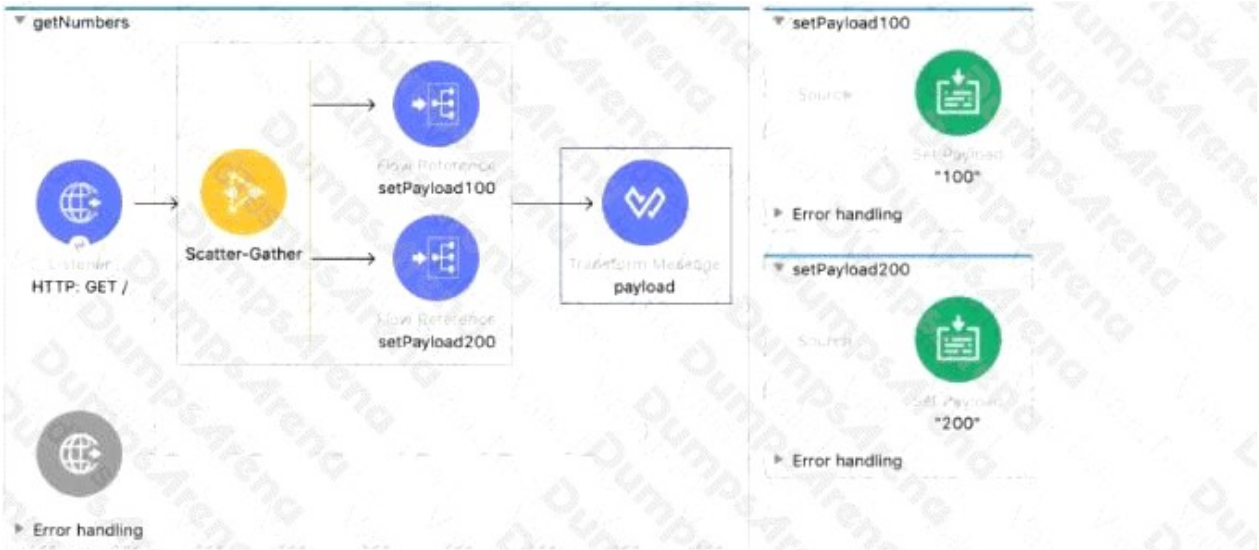
`import * from modules::MyModule` imports every publicly exposed function, variable, and type from the `MyModule.dwl` DataWeave module. DataWeave resolves modules stored below `src/main/resources` by using a double-colon-separated path that corresponds to the directory structure, without the `.dwl` extension. Therefore, a file at `src/main/resources/modules/MyModule.dwl` is referenced as `modules::MyModule`.

The wildcard import makes the module's exported members available directly in the importing script's scope. After this import, the function can be invoked by its declared name, such as `formatString("example")`, rather than through a module namespace. The import must appear in the DataWeave header, before the separator (`---`) and body expression. This approach is appropriate when the script needs access to the module's members directly and the imported names do not conflict with locally declared names or names from other imports.

MuleSoft documents DataWeave module imports, including wildcard imports, at Create and Import a Custom DataWeave Module. The DataWeave language reference also describes the import syntax for functions and modules.

QUESTION NO: 25

Refer to the exhibits.



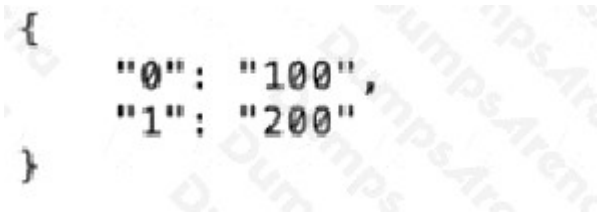

```
<flow name="getNumbers" >
    <http:listener doc:name="HTTP: GET /" config-ref="HTTP_Listener_config" path="/"/>
    <scatter-gather doc:name="Scatter-Gather" >
        <route >
            <flow-ref doc:name='setPayload100' name='setPayload100'/>
        </route>
        <route >
            <flow-ref doc:name="setPayload200" name="setPayload200"/>
        </route>
    </scatter-gather>
    <ee:transform doc:name="payload">
        <ee:message >
            <ee:set-payload ><![CDATA[%dw 2.0
            output application/json
            ---
            payload]]></ee:set-payload>
        </ee:message>
    </ee:transform>
</flow>
<flow name="setPayload100" ><set-payload value='#["100"]' doc:name='"100"' /></flow>
<flow name="setPayload200" ><set-payload value='#["200"]' doc:name='"200"' /></flow>
```

Each route in the Scatter-Gather sets the payload to the number shown in the label. Whatresponse is returned to a web client request to the HTTP Listener?

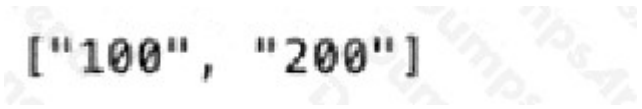
A)

```
[
    {
        "attributes": ...,
        "payload": "100"
    },
    {
        "attributes": ...,
        "payload: "200"
    }
]
```

B)



C)



D)

A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: D

Explanation:

The response represented by *Option D* is correct because a Scatter-Gather executes each configured route concurrently and then aggregates the resulting Mule events before continuing the flow. The HTTP Listener therefore returns the Scatter-Gather's combined result rather than only one route's payload. Each route result is retained as an individual event entry, indexed by its route position. Consequently, the aggregate response contains one entry for the route that set its payload to `100` and another for the route that set its payload to `200`. Each entry includes the route event's `payload` and `attributes`, producing a structure equivalent to `{ "0": { "payload": "100", "attributes": ... }, "1": { "payload": "200", "attributes": ... } }`. This behavior is important because Scatter-Gather preserves the distinct output from every successful route instead of merging their payload values into a single scalar value. Mule's Scatter-Gather documentation describes that the router collects the route results and provides the aggregated set of Mule events to the next message processor; because there is no subsequent transformation in the shown flow, that aggregate is the HTTP response. See the Mule Runtime Scatter-Gather Router documentation and the Scatter-Gather XML reference.

QUESTION NO: 26

A Utlility.dwl is located in a Mule project at src/main/resources/modules. The Utility.dwl file defines a function named encryptString that encrypts a String What is thecorrect DataWeave to call the encryptString function in a Transform Message component?

A. 1. %dw 2.0

B. %dw 2.0 output application/json import modules::Utility --- Utility::encryptString("John Smith")

C. 1. %dw 2.0

D. 1. %dw 2.0

ANSWER: B

Explanation:

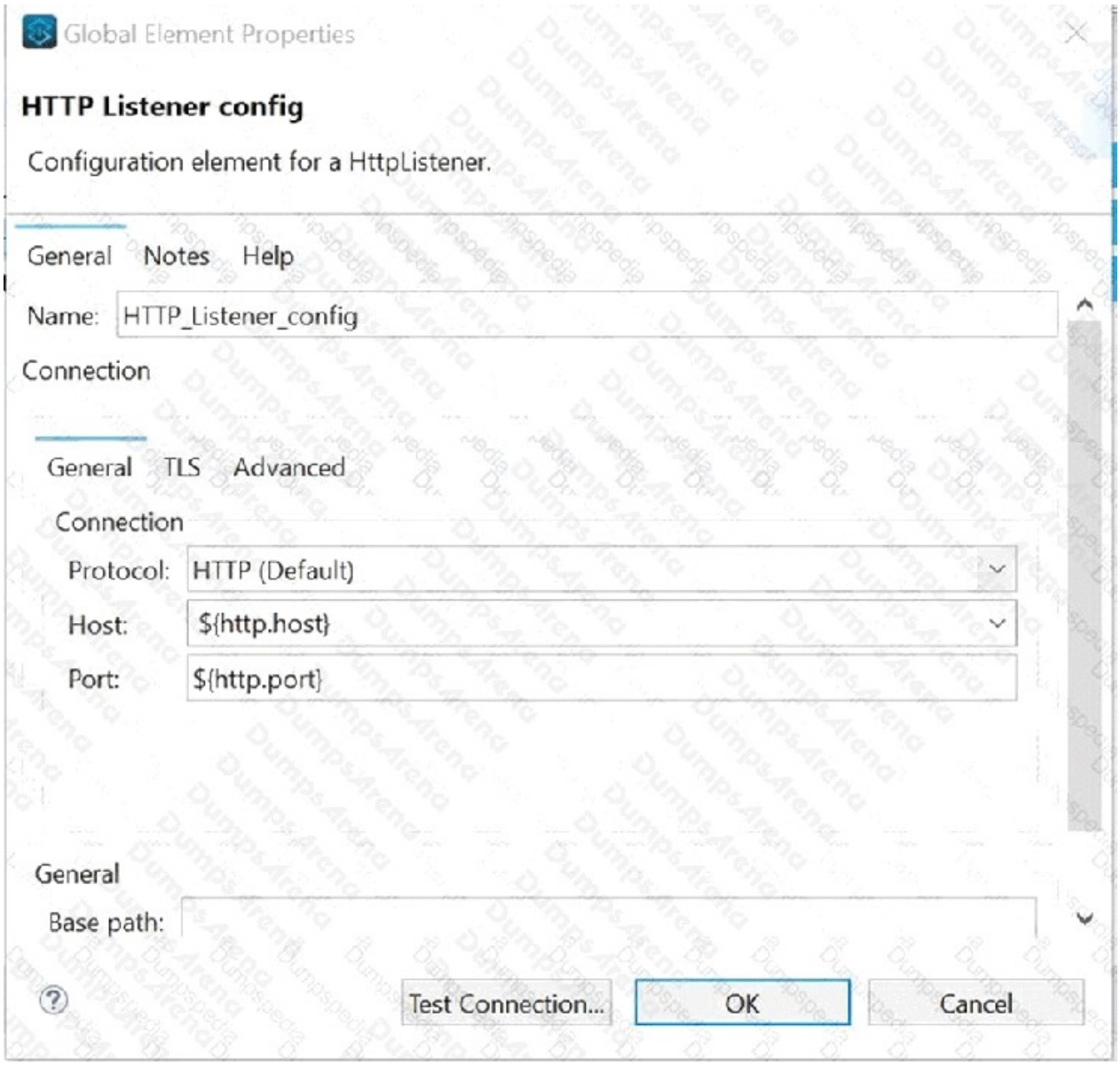
The correct DataWeave imports the custom module by its classpath-relative module name, `modules::Utility`, and then invokes the function with the module-qualified function name, `Utility::encryptString("John Smith")`. Files under `src/main/resources` are available on the application classpath. Therefore, because the file is stored at `src/main/resources/modules/Utility.dwl`, its import path is `modules::Utility`; the `.dwl` extension is not included in the import directive. Importing the module as a whole makes its exported functions available through the module namespace, represented here by `Utility`. The Transform Message script can then return the encrypted value as its output. MuleSoft documents that custom DataWeave modules can be imported using their path relative to `src/main/resources`, and functions from an imported module are called with the module name followed by `::`. See the DataWeave Modules documentation and the DataWeave language introduction for module import and function invocation syntax.

QUESTION NO: 27

Refer to the below exhibit.

A Mule applicationconfigures a property placeholder file named config.yaml to set some property placeholders for an HTTP connector.

What is the valid properties placeholder file to set these values?



A. http:
  host = "localhost"
  port = "8081"

B. http:
  basepath: "api"
  host: "localhost"
  port: "8081"

C. http.host = localhost
http.port = 8081

D. { http: basePath: "api", port: "8081", host: " localhost" }

ANSWER: B

Explanation:

The valid properties file is `http:` with the indented `basepath`, `host`, and `port` entries. A `.yaml` property file represents nested configuration as YAML mappings, so the `http` key creates a parent object and its indented values become child properties. Mule can then resolve these values through property placeholders using the corresponding dot notation, such as `${http.host}`, `${http.port}`, and `${http.basepath}`. Quoting the values is valid YAML and is especially appropriate for configuration values such as a port when the application intends to treat it consistently as a string before the HTTP connector consumes it. The indentation is essential because it expresses the hierarchy required for the property

names referenced by the connector configuration. Mule applications load externalized application properties through the Configuration Properties mechanism, which supports YAML files and nested property access. See MuleSoft's documentation on configuring application properties and its property configuration guidance.

QUESTION NO: 28

What MuleSoft API-led connectivity layer is intended to expose part of a backend database without business logic?

A. Data layer

B. Process layer

C. Experience layer

D. System layer

ANSWER: D

Explanation:

The System layer is intended to expose underlying systems of record, such as databases, ERP platforms, billing systems, and CRM systems, in a consistent and reusable way. A System API abstracts the backend system's implementation details and provides a stable interface for consumers, insulating them from changes to database schemas, vendor platforms, connection methods, or legacy-system complexity. When the requirement is to expose data from a backend database without applying business logic, it aligns directly with the System layer's responsibility: securely unlock and provide access to the source system's data and capabilities.

In API-led connectivity, System APIs form the foundational layer. They are designed around the systems they expose rather than around a business process or a specific consuming channel. This separation enables downstream APIs and applications to reuse the database access interface without becoming coupled to the database itself. MuleSoft describes System APIs as APIs that expose and provide access to core systems of record, while higher layers compose data into business processes or tailor it to consumer needs. See MuleSoft's API-led connectivity overview and its API-led connectivity documentation.