DUMPSBOSS.

MuleSoft Certified Developer - Level 1 (Mule 4)

Mulesoft MuleSoft-Certified-Developer-Level-1

Version Demo

Total Demo Questions: 11

Total Premium Questions: 118

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Mule 4 Fundamentals	41
Topic 2, DataWeave	24
Topic 3, Building Mule Applications	24
Topic 4, Error Handling	8
Topic 5, Deployment and Management	21
Total	118

QUESTION NO: 1

What is the output type of the DataWeave map operator?

A. Object

B. Array

C. Map

D. String

ANSWER: B

Explanation:

The correct answer is **Array**. In DataWeave, `map` is an array transformation operator: it iterates over each element of an input array, evaluates the mapping expression for that element, and collects every resulting value into a new array. Its documented type signature expresses this directly: given an `Array<T>` and a mapper that converts each item to a value of type `R`, the result is `Array<R>`. The mapper can change the type and structure of each individual element—for example, it can convert strings to numbers or input objects to differently shaped objects—but the enclosing result produced by `map` remains an array.

For example, applying `payload map ($.name)` to an array of customer objects returns an array containing the extracted names. Similarly, mapping numeric values with an arithmetic expression returns an array of the calculated values. This predictable collection-preserving behavior makes `map` the standard choice when a flow needs to transform every item in an array. MuleSoft documents the operator and examples in the DataWeave map function reference and the Map DataWeave cookbook.

QUESTION NO: 2

A company 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 PO5555?

A. /orders/{PO5555}

B. /orders/order=PO5555

C. /orders?order=PO5555

D. /orders/PO5555

ANSWER: D

Explanation:

The URI `/orders/PO5555` is correct because a purchase order ID uniquely identifies one specific purchase-order resource. In RAML, this is modeled using a URI parameter in the resource path, such as `/orders/{orderId}`. The braces are part of the RAML API definition and declare a variable segment; a client does not send the braces literally. Instead, it substitutes the actual identifier value when making the request, producing `/orders/PO5555`.

Using a path parameter for a unique resource identifier follows RESTful resource-naming practices: the collection is represented by `/orders`, while an individual member of that collection is represented by appending its identifier. This makes the request unambiguous and maps naturally to a RAML resource declaration with a nested `/{orderId}` path. MuleSoft's API design guidance describes URI parameters as the appropriate mechanism for identifying a particular resource within a collection. See the MuleSoft RAML design guidance and the RAML URI parameters specification.

QUESTION NO: 3

A flow needs to combine and return data from two different data sources. It contains a Database SELECT operation followed by an HTTP Request operation.

What is the method to capture both payloads so the payload from the second request does not overwrite that from the first?

A. Save the payload from the Database SELECT operation to a variable.

B. Put the Database SELECT operation inside a Cache scope

C. Put the Database SELECT operation inside a Message Enricher scope

D. Nothing, previous payloads are combined into the next payload

ANSWER: A

Explanation:

Save the payload from the Database SELECT operation to a variable. In Mule 4, each event has one current payload, and message processors commonly replace that payload with their own output. A Database SELECT operation sets the payload to the query result, while the subsequent HTTP Request sets it to the HTTP response body. To retain the database result before making the HTTP call, use a Set Variable component immediately after the Database SELECT operation, assigning the current payload to a flow variable such as `dbResult`. The HTTP Request can then run normally and place its response in the current payload. Afterward, a Transform Message component can construct the required combined response by accessing the saved database value through `vars.dbResult` and the HTTP response through `payload`. Flow variables are designed for exactly this type of per-event, in-flow state: they travel with the Mule event through the flow and can be referenced by later components. This preserves both source results without changing the request behavior or losing the second operation's response. See MuleSoft's documentation on Set Variable and Mule event structure.

QUESTION NO: 4

Refer to the exhibit.

```
#%RAML 1.0
title: Accounts API
version: 1.0

/accounts:
  get:
    description: Get all accounts
    responses:
      200:
        body:
          application/json:
            example:
              id: "48292"
              name: Geordi La Forge
              address: 1 Forge Way, Midgard, CA 95928
              customer_since: "2014-01-04"
              balance: 4829.29
  post:
    description: Create an account
    body:
      application/json:
        example:
          name: Geordi La Forge
          address: 1 Forge Way, Midgard, CA 95928
          customer_since: "2014-01-04"
          balance: 4829.29
          bank_agent_id: "48-SJT-282924-KL"
```

What data is expected by the POST /accounts endpoint?

A.
```
<item>
    <id>48292</id>
    <name>Geordi La Forge</name>
    <address>1 Forge Way, Midgard, CA 95928</address>
    <customer_since>2014-01-04</customer_since>
    <balance>4829.29</balance>
</item>
```

B.
```
<item>
    <name>Geordi La Forge</name>
    <address>1 Forge Way, Midgard, CA 95928</address>
    <customer_since>2014-01-04</customer_since>
    <balance>4829.29</balance>
    <bank_agent_id>48-SJT-282924-KL</bank_agent_id>
</item>
```

C.
```
{
    "name": "Geordi La Forge",
    "address": "1 Forge Way, Midgard, CA 95928",
    "customer_since": "2014-01-04",
    "balance": 4829.29,
    "bank_agent_id": "48-SJT-282924-KL"
}
```

D.
```
{
    "id": "48292",
    "name": "Geordi La Forge",
    "address": "1 Forge Way, Midgard, CA 95928",
    "customer_since": "2014-01-04",
    "balance": 4829.29
}
```

A. Option A

B. Option B

C. Option C

D. Option D

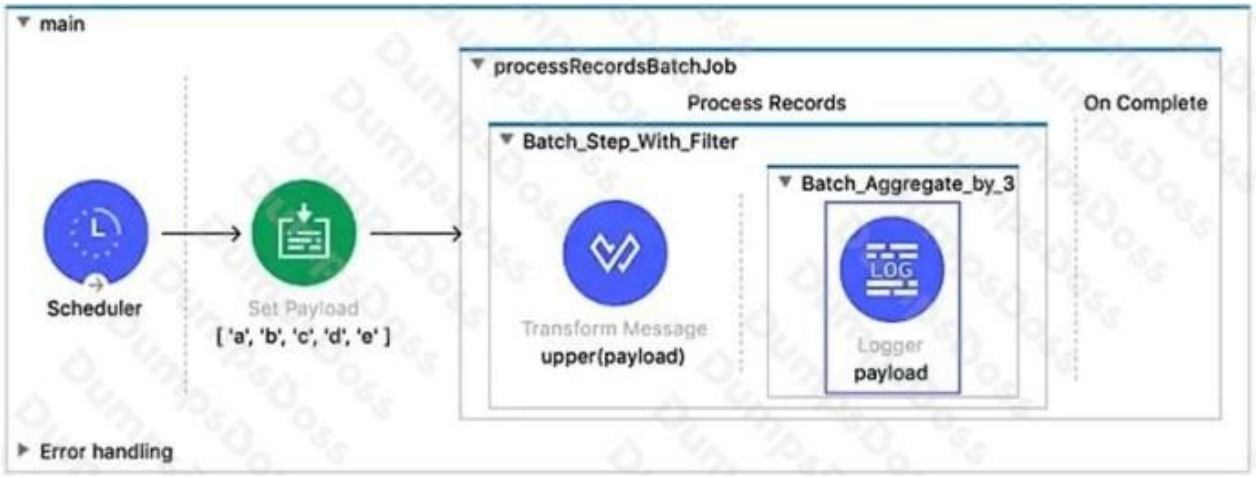
ANSWER: A

Explanation:

Option A is correct. For a RAML-defined POST operation, the data expected by the endpoint is determined by the operation's `body` declaration, including its media type and the schema or RAML data type assigned to that body. A client invoking `POST /accounts` must therefore send a request entity that conforms to the account payload structure shown in the API contract: it uses the declared content type and supplies the fields with the value types, nesting, and requiredness defined by that body type. This contract-first definition lets Mule applications and API consumers consistently validate and process incoming account-creation requests. In RAML, a resource method can declare one or more request-body media types, and each media type can reference a type that specifies the JSON object shape expected at runtime. MuleSoft's API specifications use RAML to document these request and response contracts for consumers and implementations. See MuleSoft's RAML API design documentation and the RAML 1.0 body declaration specification.

QUESTION NO: 5

Refer to the exhibits.




```
<flow name="main" >
    <scheduler doc:name="Scheduler" > <scheduling-strategy >
    <fixed-frequency frequency="10000"/> </scheduling-strategy> </scheduler>
    <set-payload value="#[[ 'a', 'b', 'c', 'd', 'e' ]]" doc:name="[ 'a', 'b', 'c', 'd', 'e' ]" />
    <batch:job jobName="processRecordsBatchJob" >
        <batch:process-records >
            <batch:step name="Batch_Step_With_Filter"
                acceptExpression='#[not (payload contains "b") ]'>
                <ee:transform doc:name="upper(payload)"> <ee:message >
                        <ee:set-payload ><![CDATA[%dw 2.0
                            output application/json
                            ---
                            upper(payload)]]></ee:set-payload>
                    </ee:message></ee:transform>
                <batch:aggregator doc:name="Batch_Aggregate_by_3" size="3">
                    <logger level="INFO" doc:name="payload"
                        message="#[output application/json --- payload]"/>
                </batch:aggregator>
            </batch:step>
        </batch:process-records>
    </batch:job>
</flow>
```

The input array of strings is processed by the batch job that processes, filters, and aggregates the values.

What is the last message logged by the Logger component after the batch job completes processing?

A. [[“A”, “C”, “D”], [“E”]]

B. [“E”]

C. [“D”, “E”]

D. [“A”, “C”, “D”, “E”]

ANSWER: D

Explanation:

`[ "A", "C", "D", "E" ]` is the message logged after the batch job completes. The batch job first creates records from the input collection and processes each record independently. The filtering step excludes the record whose value is `"B"`, so that record does not continue as a successfully processed record. The records with values `"A"`, `"C"`, `"D"`, and `"E"` proceed through the batch processing and aggregation logic.

Aggregation controls how processed records are grouped while the aggregation operation runs; it does not cause the final completed-job result logged after processing to remain nested according to aggregation blocks. At the completion point, the successful record values are represented as the resulting collection, preserving the processed records that passed the filter. Therefore, the final Logger output is the flat array containing the four retained values in their processing order. Mule batch processing separates record loading, per-record processing, and completion handling, which is why the result is evaluated only after all eligible records and aggregation work have finished. See MuleSoft's Batch Processing concept documentation and its Batch Aggregator documentation.

QUESTION NO: 6

Why would a Mule application use the ${http.port} property placeholder for its HTTP Listener port when it is deployed to CloudHub?

A. Allows CloudHub to automatically change the HTTP port to allow external clients to connect to the HTTP Listener

B. Allows clients to VPN directly to the application at the Mule application's configured HTTP port

C. Allows MuleSoft Support to troubleshoot the application by connecting directly to the HTTP Listener

D. Allows CloudHub to automatically register the application with API Manager

ANSWER: A

Explanation:

Allows CloudHub to automatically change the HTTP port to allow external clients to connect to the HTTP Listener. CloudHub places an application behind its managed load balancer and exposes a public application endpoint. Incoming client traffic reaches that endpoint over the platform-managed HTTP or HTTPS ports, then CloudHub routes the request to the port on which the Mule application is listening inside its worker. Using `${http.port}` makes the listener's internal port configurable through an application property rather than hard-coding an environment-specific value in the Mule configuration.

This is important because the same deployable application can be run in different environments while each environment supplies the appropriate listener-port value. In CloudHub deployments, the property convention supports the platform's routing model and lets CloudHub direct externally received requests to the application listener correctly. The public URL is therefore decoupled from the internal listener-port configuration, while the application remains portable across local development, testing, and CloudHub deployment. MuleSoft documents CloudHub deployment and networking behavior in the CloudHub deployment documentation; HTTP Listener configuration and property-based values are documented in the HTTP Listener reference.

QUESTION NO: 7

Refer to the exhibits.

```
    <http:request method="POST" doc:name="POST HTTP /child" url="http://localhost:8081/child">
    </http:request>
    <logger level="INFO" doc:name="Logger"/>
</flow>

<flow name="child"  >
    <http:listener doc:name="HTTP POST /child"  config-ref="HTTP_Listener_config" path="/child"/>
```

The main flow contains an HTTP Request in the middle of the flow. The HTTP Listeners and HTTP Request use default configurations.

A web client sends a GET request to the main flow's HTTP Listener. The GET request includes query for the pedigree of a piano.

What values are accessible to the Logger component at the end of the main flow?

A. payload

B. payload pedigree query params

C. payload producer var

D. payload pedigree query params producer var

ANSWER: C

Explanation:

`payload producer var` is accessible at the Logger because an HTTP Request operation produces a new message from the HTTP response. After the request completes, the event payload is the body returned by the target HTTP service, so the Logger can access that response payload. Mule variables belong to the Mule event and continue through the flow unless a component explicitly removes or changes them. Therefore, the variable holding the producer remains available after the HTTP Request.

The inbound HTTP Listener initially creates request attributes that include the client request's query parameters, such as the piano pedigree query. However, the HTTP Request replaces the message attributes with attributes describing its HTTP response. At the end of the main flow, the current attributes no longer represent the original listener request, while the response payload and the preserved producer variable remain available. This behavior follows Mule's event model: the payload and attributes represent the current message, whereas variables retain flow state independently. See MuleSoft's Mule event structure documentation and the HTTP Connector documentation.

QUESTION NO: 8

What are the features of CloudHub Fabric?

A. Non-persistent queue

B. Horizontal Scaling

C. VPN Mock Services

D. None of these

ANSWER: B

Explanation:

Horizontal Scaling is a CloudHub Fabric capability. CloudHub Fabric lets organizations run Mule applications on Kubernetes infrastructure that they manage while using Anypoint Runtime Manager for application deployment and operational control. Because an application is deployed as one or more replicas in the Kubernetes-based fabric, its capacity can be increased horizontally by scaling the number of replicas. This enables the application to distribute processing across multiple running instances and supports changing workload demands without requiring a redesign of the integration application.

In practice, horizontal scaling is particularly useful for APIs and integrations with variable or high request volumes. The platform's deployment model uses Kubernetes orchestration to maintain the requested replica count and provide the underlying container scheduling and health-management behavior. Application-level design must still account for state, external dependencies, and idempotency when replicas process requests concurrently, but replica scaling is the relevant CloudHub Fabric feature identified here. MuleSoft documents CloudHub Fabric as a Kubernetes-based runtime-plane deployment option and describes deploying and managing applications on it through Runtime Manager. See CloudHub Fabric documentation and Deploying applications to CloudHub Fabric.

QUESTION NO: 9

Refer to the exhibits.

```
<flow name="logPayload" >
    <scheduler doc:name="Scheduler" >
        <scheduling-strategy >
            <fixed-frequency />
        </scheduling-strategy>
    </scheduler>
    <set-payload value="#[{'year': '2020'}]" doc:name="{'year': '2020'}" />
    <logger level="INFO" doc:name="Logger" message=???? />
</flow>
```

The Set Payload transformer's value is set to {'year': '2020'}.

What message value should be added to the Logger component to output the message 'The year is '2020', without hardcoding 2020?

A. 'The year is #[payload.year]'

B. '#[The year is $(payload.year)]'

C. '#["The year is ++ payload.year"]'

D. '#["The year is " + payload.year]'

ANSWER: A

Explanation:

'The year is #[payload.year]' is the correct Logger message value because Mule supports embedding a DataWeave expression inside literal message text with the `#[...]` expression delimiter. The literal portion, `The year is ` , is written directly in the Logger message field, while `payload.year` dynamically reads the `year` field from the current payload. Since the Set Payload operation produces an object whose `year` value is the string `2020`, evaluating that selector yields `2020`. The Logger therefore emits the combined text `The year is 2020` at runtime, without putting the year value directly into the Logger configuration. This approach keeps the message tied to the payload state, so a later payload containing a different year would automatically produce the corresponding output. Mule Logger message values accept DataWeave expressions, and DataWeave object selectors use dot notation to access fields such as `payload.year`. See the Logger Component reference and the DataWeave selectors documentation.

QUESTION NO: 10

A Database On Table Row listener retrieves data from a CUSTOMER table that contains a primary key user_id column and an increasing login_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 login_date_time column

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

C. Set the target value to the last retrieved user_id value

D. Set the watermark column to the user_id column

ANSWER: A

Explanation:

Set the watermark column to the login_date_time column. The Database On Table Row source uses a watermark to maintain its position between polling executions. After processing records, Mule persists the highest observed value for the configured watermark column and uses that value in subsequent polls to identify rows added after the prior watermark. This allows the listener to progress through the table without rereading rows that it has already retrieved.

A suitable watermark column must provide an ordered, reliably advancing value so that newly available rows can be distinguished from previously retrieved rows. In this scenario, `login_date_time` is explicitly described as increasing and non-duplicated, making it an appropriate cursor for the listener. Configuring it as the watermark ensures that every processed row has a distinct position in the polling sequence and that later polls continue after the last retrieved timestamp. Mule manages the stored watermark state for the source, rather than requiring the flow to manually retain the prior retrieved value.

See the MuleSoft Database Connector documentation for the On Table Row source and watermark behavior: Database Connector Documentation and Database On Table Row Source.

QUESTION NO: 11

An API has been created in Design Center.

What is the next step to make the API discoverable?

A. Deploy the API to a Maven repository.

B. Publish the API from inside flow designer.

C. Publish the API to Anypoint Exchange.

D. Enable autodiscovery in API Manager.

ANSWER: C

Explanation:

Publishing the API to Anypoint Exchange is the appropriate next step because Exchange is MuleSoft's catalog for sharing and discovering reusable assets, including API specifications, API fragments, examples, templates, connectors, and other integration assets. Once an API specification created in Design Center is published to Exchange, authorized users in the relevant Anypoint organization or business group can search for it, view its documentation and metadata, and reuse it when designing or implementing integrations. Publishing also establishes the API asset's version in the catalog, helping teams govern and communicate the API contract across its lifecycle. Design Center supports publishing an API specification directly to Exchange, where it becomes available as a managed asset rather than remaining only in the design workspace. The API's visibility is governed by the organization and any applicable permissions, so users must have access to the relevant Exchange content to discover it. See MuleSoft's Anypoint Exchange documentation for the purpose and use of Exchange, and its Design Center API design documentation for information on working with API specifications and publishing assets.