Free Questions for MCIA-Level-1 by actualtestdumps

Shared by Bolton on 07-06-2022

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

A popular retailer is designing a public API for its numerous business partners. Each business partner will invoke the API at the URL 58. https://api.acme.com/partnefs/vl. The API implementation is estimated to require deployment to 5 CloudHub workers.

The retailer has obtained a public X.509 certificate for the name apl.acme.com, signed by a reputable CA, to be used as the server certificate.

Where and how should the X.509 certificate and Mule applications be used to configure load balancing among the 5 CloudHub workers, and what DNS entries should be configured in order for the retailer to support its numerous business partners?

Options:

A) Add the X.509 certificate to the Mule application's deployable archive, then configure a CloudHub Dedicated Load Balancer (DLB) for each of the Mule application's CloudHub workers
Create a CNAME for api.acme.com pointing to the DLB's A record

B) Add the X.509 certificate to the CloudHub Shared Load Balancer (SLB), not to the Mule application
Create a CNAME for api.acme.com pointing to the SLB's A record

C) Add the X.509 certificate to a CloudHub Dedicated Load Balancer (DLB), not to the Mule application
Create a CNAME for api.acme.com pointing to the DLB's A record

D) Add the x.509 certificate to the Mule application's deployable archive, then configure the CloudHub Shared Load Balancer (SLB) for each of the Mule application's CloudHub workers

Create a CNAME for api.acme.com pointing to the SLB's A record

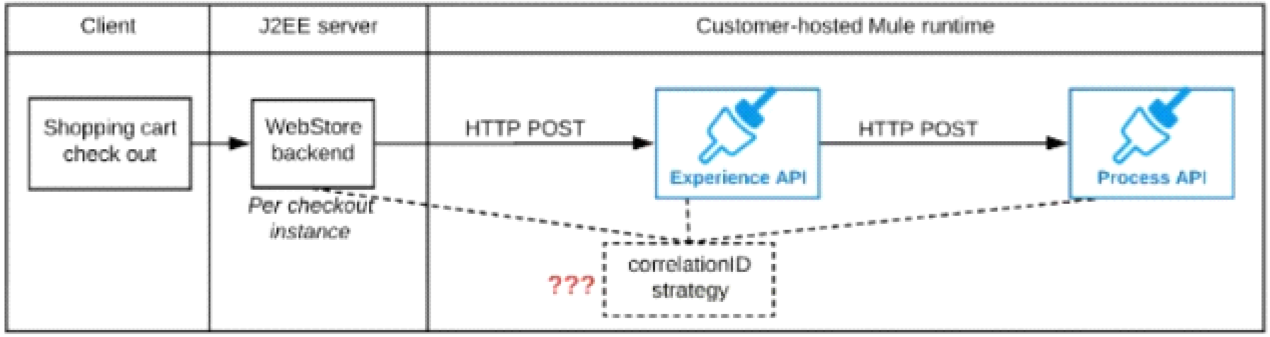
Answer:

C

Question 2

Question Type: **MultipleChoice**

Refer to the exhibit.



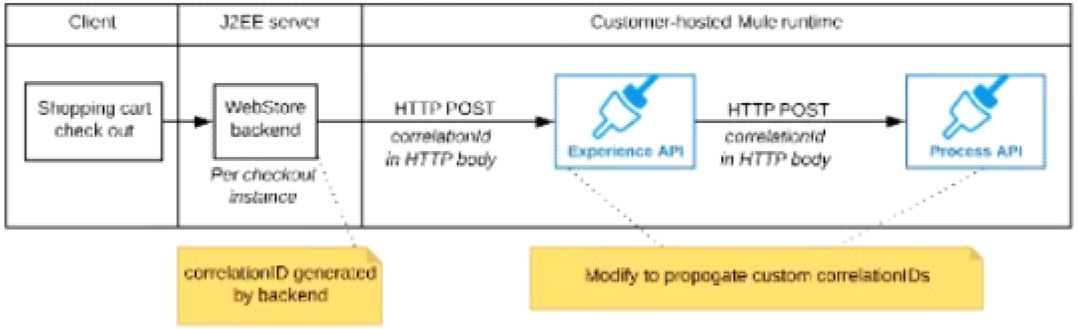
A shopping cart checkout process consists of a web store backend sending a sequence of API invocations to an Experience API, which in turn invokes a Process API. All API invocations are over HTTPS POST. The Java web store backend executes in a Java EE application server, while all API implementations are Mule applications executing in a customer -hosted Mule runtime.

End-to-end correlation of all HTTP requests and responses belonging to each individual checkout Instance is required. This is to be done through a common correlation ID, so that all log entries written by the web store backend, Experience API implementation, and Process API implementation include the same correlation ID for all requests and responses belonging to the same checkout instance.

What is the most efficient way (using the least amount of custom coding or configuration) for the web store backend and the implementations of the Experience API and Process API to participate in end-to-end correlation of the API invocations for each checkout instance?

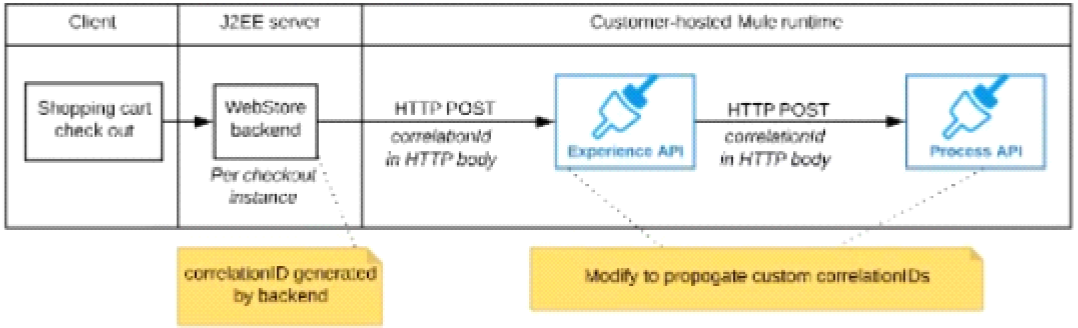
Options:

A) The web store backend, being a Java EE application, automatically makes use of the thread-local correlation ID generated by the Java EE application server and automatically transmits that to the Experience API using HTTP-standard headers
No special code or configuration is included in the web store backend, Experience API, and Process API implementations to generate and manage the correlation ID


Client
J2EE server
Customer-hosted Mule runtime
Shopping cart check out
WebStore backend
Per checkout instance
HTTP POST
correlationId in HTTP body
Experience API
HTTP POST
correlationId in HTTP body
Process API
correlationID generated by backend
Modify to propogate custom correlationIDs

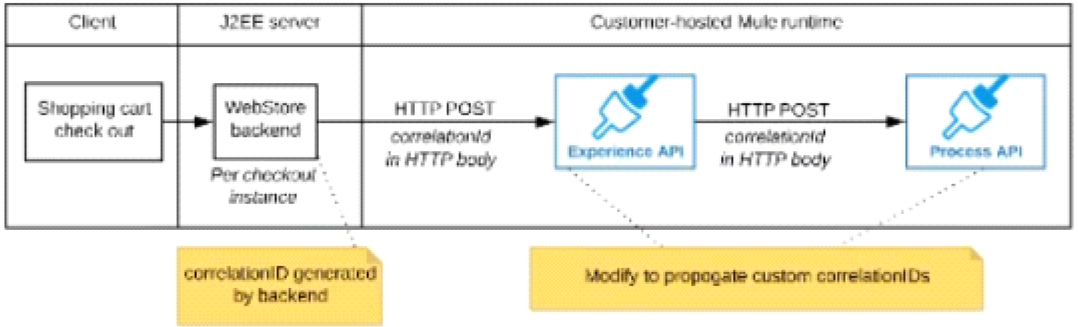
B) The web store backend generates a new correlation ID value at the start of checkout and sets it on the X-CORRELATION-lt HTTP request header In each API invocation belonging to that checkout

No special code or configuration is included in the Experience API and Process API implementations to generate and manage the correlation ID


Client
J2EE server
Customer-hosted Mule runtime
Shopping cart check out
WebStore backend
Per checkout instance
HTTP POST
correlationId in HTTP body
Experience API
HTTP POST
correlationId in HTTP body
Process API
correlationID generated by backend
Modify to propogate custom correlationIDs

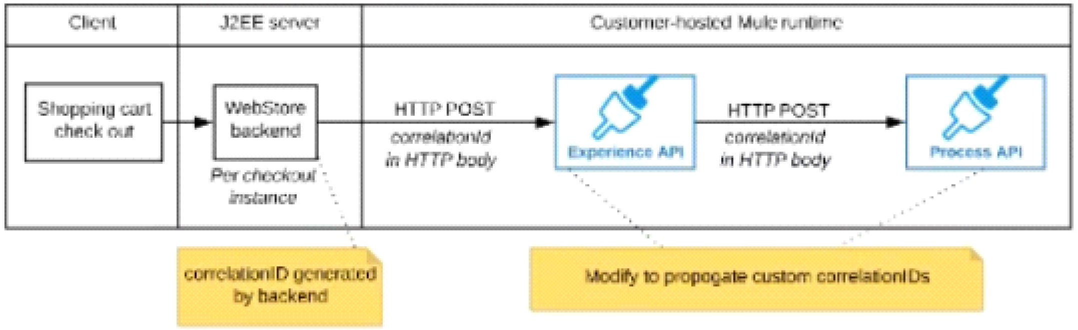
C) The Experience API implementation generates a correlation ID for each incoming HTTP request and passes it to the web store backend in the HTTP response, which includes it in all subsequent API invocations to the Experience API.

The Experience API implementation must be coded to also propagate the correlation ID to the Process API in a suitable HTTP request header


Client
J2EE server
Customer-hosted Mule runtime
Shopping cart check out
WebStore backend
Per checkout instance
HTTP POST
correlationId in HTTP body
Experience API
HTTP POST
correlationId in HTTP body
Process API
correlationID generated by backend
Modify to propogate custom correlationIDs

D) The web store backend sends a correlation ID value in the HTTP request body In the way required by the Experience API

The Experience API and Process API implementations must be coded to receive the custom correlation ID In the HTTP requests and propagate It in suitable HTTP request headers


Client
J2EE server
Customer-hosted Mule runtime
Shopping cart check out
WebStore backend
Per checkout instance
HTTP POST
correlationId in HTTP body
Experience API
HTTP POST
correlationId in HTTP body
Process API
correlationID generated by backend
Modify to propogate custom correlationIDs

Answer:

B

Question 3

Question Type: MultipleChoice

A popular retailer is designing a public API for its numerous business partners. Each business partner will invoke the API at the URL 58. https://api.acme.com/partnefs/vl. The API implementation is estimated to require deployment to 5 CloudHub workers.

The retailer has obtained a public X.509 certificate for the name apl.acme.com, signed by a reputable CA, to be used as the server certificate.

Where and how should the X.509 certificate and Mule applications be used to configure load balancing among the 5 CloudHub workers, and what DNS entries should be configured in order for the retailer to support its numerous business partners?

Options:

A) Add the X.509 certificate to the Mule application's deployable archive, then configure a CloudHub Dedicated Load Balancer (DLB) for each of the Mule application's CloudHub workers
Create a CNAME for api.acme.com pointing to the DLB's A record

B) Add the X.509 certificate to the CloudHub Shared Load Balancer (SLB), not to the Mule application
Create a CNAME for api.acme.com pointing to the SLB's A record

C) Add the X.509 certificate to a CloudHub Dedicated Load Balancer (DLB), not to the Mule application
Create a CNAME for api.acme.com pointing to the DLB's A record

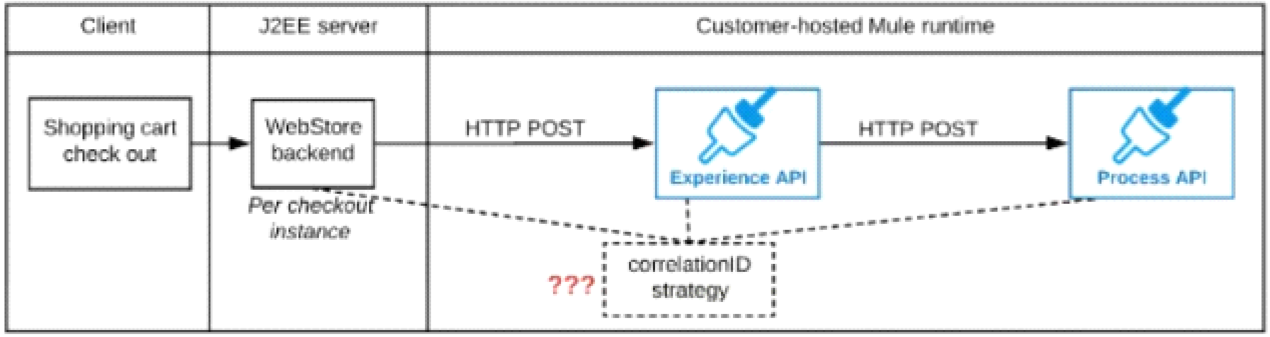
D) Add the x.509 certificate to the Mule application's deployable archive, then configure the CloudHub Shared Load Balancer (SLB) for each of the Mule application's CloudHub workers
Create a CNAME for api.acme.com pointing to the SLB's A record

Answer:

C

Question 4

Refer to the exhibit.



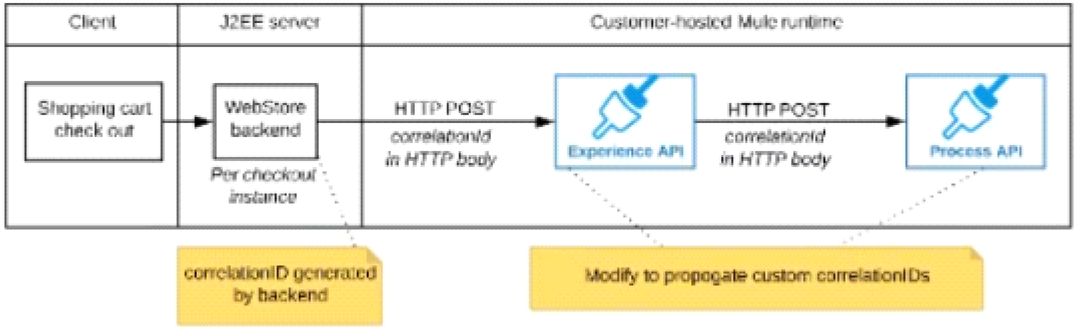
A shopping cart checkout process consists of a web store backend sending a sequence of API invocations to an Experience API, which in turn invokes a Process API. All API invocations are over HTTPS POST. The Java web store backend executes in a Java EE application server, while all API implementations are Mule applications executing in a customer -hosted Mule runtime.

End-to-end correlation of all HTTP requests and responses belonging to each individual checkout Instance is required. This is to be done through a common correlation ID, so that all log entries written by the web store backend, Experience API implementation, and Process API implementation include the same correlation ID for all requests and responses belonging to the same checkout instance.

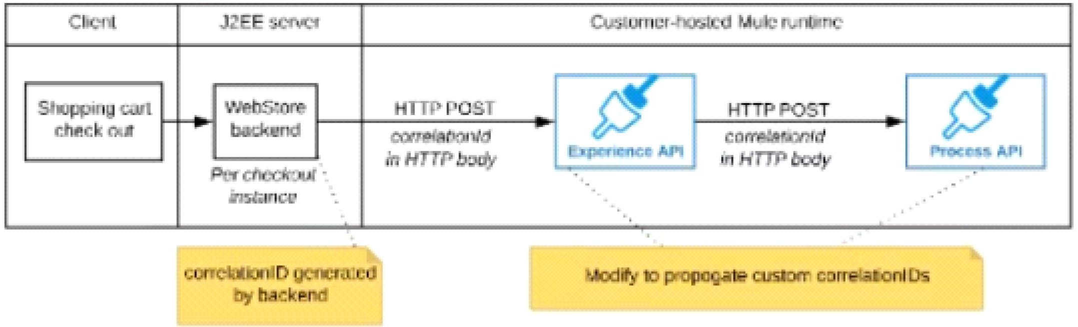
What is the most efficient way (using the least amount of custom coding or configuration) for the web store backend and the implementations of the Experience API and Process API to participate in end-to-end correlation of the API invocations for each checkout instance?

Options:

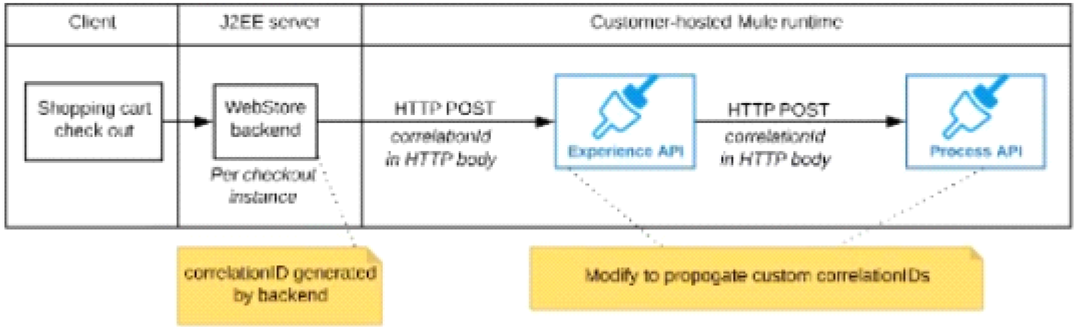
A) The web store backend, being a Java EE application, automatically makes use of the thread-local correlation ID generated by the Java EE application server and automatically transmits that to the Experience API using HTTP-standard headers
No special code or configuration is included in the web store backend, Experience API, and Process API implementations to generate and manage the correlation ID


Client
J2EE server
Customer-hosted Mule runtime
Shopping cart check out
WebStore backend
Per checkout instance
HTTP POST
correlationId in HTTP body
Experience API
HTTP POST
correlationId in HTTP body
Process API
correlationID generated by backend
Modify to propogate custom correlationIDs

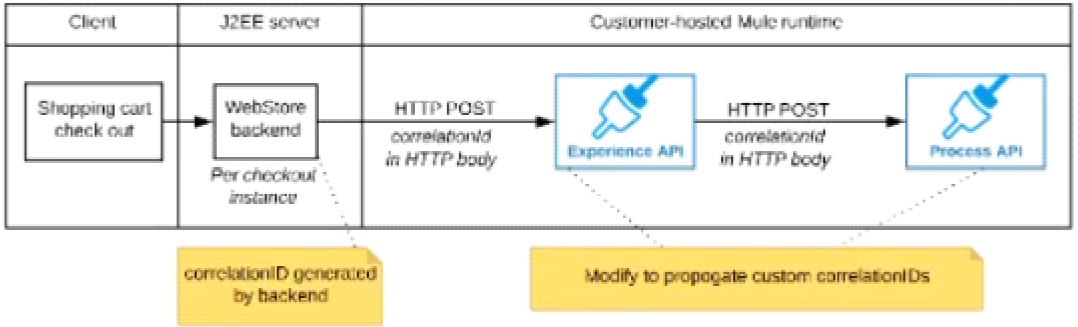
B) The web store backend generates a new correlation ID value at the start of checkout and sets it on the X-CORRELATION-lt HTTP request header In each API invocation belonging to that checkout
No special code or configuration is included in the Experience API and Process API implementations to generate and manage the correlation ID


Client
J2EE server
Customer-hosted Mule runtime
Shopping cart check out
WebStore backend
Per checkout instance
HTTP POST
correlationId in HTTP body
Experience API
HTTP POST
correlationId in HTTP body
Process API
correlationID generated by backend
Modify to propogate custom correlationIDs

C) The Experience API implementation generates a correlation ID for each incoming HTTP request and passes it to the web store backend in the HTTP response, which includes it in all subsequent API invocations to the Experience API.
The Experience API implementation must be coded to also propagate the correlation ID to the Process API in a suitable HTTP request header


Client
J2EE server
Customer-hosted Mule runtime
Shopping cart check out
WebStore backend
Per checkout instance
HTTP POST
correlationId in HTTP body
Experience API
HTTP POST
correlationId in HTTP body
Process API
correlationID generated by backend
Modify to propogate custom correlationIDs

D) The web store backend sends a correlation ID value in the HTTP request body In the way required by the Experience API
The Experience API and Process API implementations must be coded to receive the custom correlation ID In the HTTP requests and propagate It in suitable HTTP request headers


Client
J2EE server
Customer-hosted Mule runtime
Shopping cart check out
WebStore backend
Per checkout instance
HTTP POST
correlationId in HTTP body
Experience API
HTTP POST
correlationId in HTTP body
Process API
correlationID generated by backend
Modify to propogate custom correlationIDs

Answer:

B