Version: 15.0

Question: 1

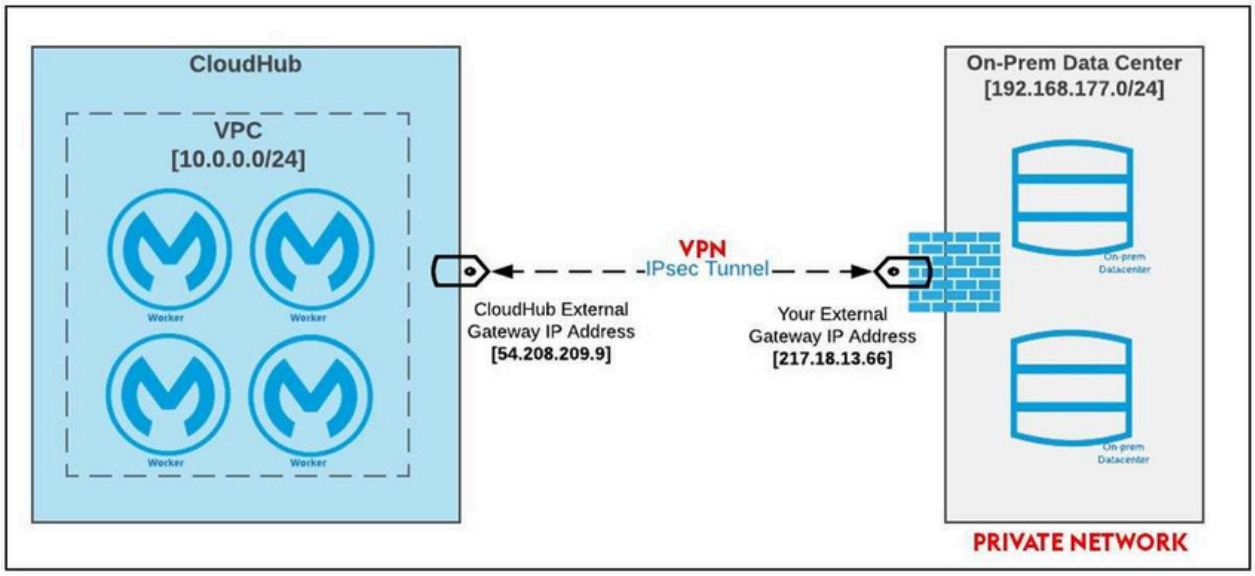
Mule applications need to be deployed to CloudHub so they can access on-premises database systems. These systems store sensitive and hence tightly protected data, so are not accessible over the internet.
What network architecture supports this requirement?

A. An Anypoint VPC connected to the on-premises network using an IPsec tunnel or AWS DirectConnect, plus matching firewall rules in the VPC and on-premises network
B. Static IP addresses for the Mule applications deployed to the CloudHub Shared Worker Cloud, plus matching firewall rules and IP
whitelisting in the on-premises network
C. An Anypoint VPC with one Dedicated Load Balancer fronting each on-premises database system, plus matching IP whitelisting in the load balancer and firewall rules in the VPC and on-premises network
D. Relocation of the database systems to a DMZ in the on-premises network, with Mule applications deployed to the CloudHub Shared Worker Cloud connecting only to the DMZ

Answer: A

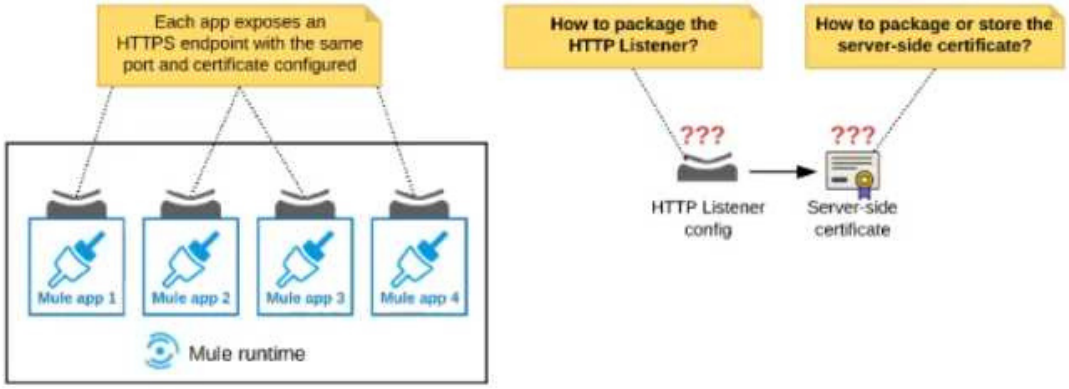
E xpl anati on:

* "Relocation of the database systems to a DMZ in the on-premises network, with Mule applications deployed to the CloudHub Shared Worker Cloud connecting only to the DMZ" is not a feasible option
* "Static IP addresses for the Mule applications deployed to the CloudHub Shared Worker Cloud, plus matching firewall rules and IP whitelisting in the on-premises network" - It is risk for sensitive data. - Even if you whitelist the database IP on your app, your app wont be able to connect to the database so this is also not a feasible option
* "An Anypoint VPC with one Dedicated Load Balancer fronting each on-premises database system, plus matching IP whitelisting in the load balancer and firewall rules in the VPC and on-premises network" Adding one VPC with a DLB for each backend system also makes no sense, is way too much work. Why would you add a LB for one system.
* Correct answer: "An Anypoint VPC connected to the on-premises network using an IPsec tunnel or AWS DirectConnect, plus matching firewall rules in the VPC and on-premises network"
IPsec Tunnel You can use an IPsec tunnel with network-to-network configuration to connect your on-premises data centers to your Anypoint VPC. An IPsec VPN tunnel is generally the recommended solution for VPC to on-premises connectivity, as it provides a standardized, secure way to connect. This method also integrates well with existing IT infrastructure such as routers and appliances.
Reference: https://docs.mulesoft.com/runtime-manager/vpc-connectivity-methods-concept



Question: 2

Refer to the exhibit.



An organization deploys multiple Mule applications to the same customer -hosted Mule runtime. Many of these Mule applications must expose an HTTPS endpoint on the same port using a server-side certificate that rotates often.
What is the most effective way to package the HTTP Listener and package or store the server-side certificate when deploying these Mule applications, so the disruption caused by certificate rotation is minimized?

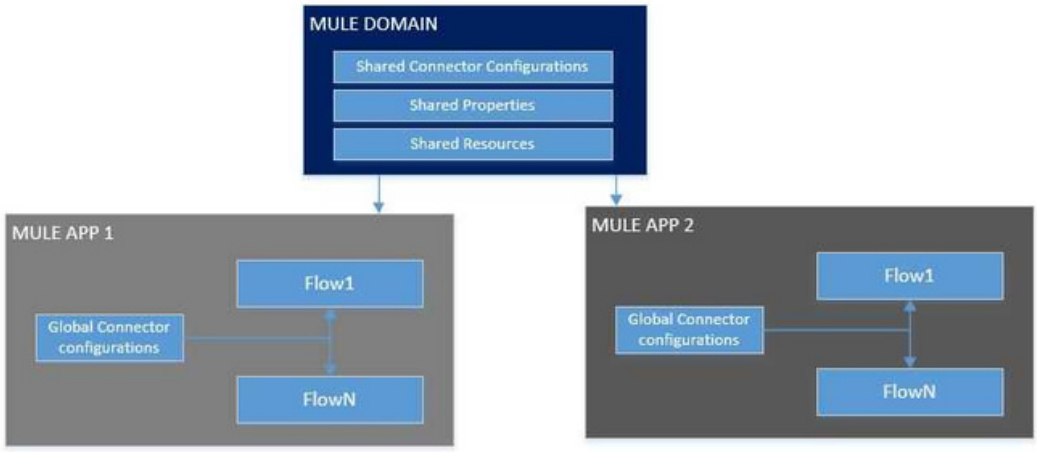
A. Package the HTTPS Listener configuration in a Mule DOMAIN project, referencing it from all Mule applications that need to expose an HTTPS endpoint Package the server-side certificate in ALL Mule APPLICATIONS that need to expose an HTTPS endpoint

B. Package the HTTPS Listener configuration in a Mule DOMAIN project, referencing it from all Mule applications that need to expose an HTTPS endpoint. Store the server-side certificate in a shared filesystem location in the Mule runtime's classpath, OUTSIDE the Mule DOMAIN or any Mule APPLICATION
C. Package an HTTPS Listener configuration In all Mule APPLICATIONS that need to expose an HTTPS endpoint Package the server-side certificate in a NEW Mule DOMAIN project
D. Package the HTTPS Listener configuration in a Mule DOMAIN project, referencing It from all Mule applications that need to expose an HTTPS endpoint. Package the server-side certificate in the SAME Mule DOMAIN project Go to Set

Answer: B

E xpl anati on:

In this scenario, both A & C will work, but A is better as it does not require repackage to the domain project at all.
Correct answer is Package the HTTPS Listener configuration in a Mule DOMAIN project, referencing it from all Mule applications that need to expose an HTTPS endpoint. Store the server-side certificate in a shared filesystem location in the Mule runtime’s classpath, OUTSIDE the Mule DOMAIN or any Mule APPLICATION.
What is Mule Domain Project?
* A Mule Domain Project is implemented to configure the resources that are shared among different projects. These resources can be used by all the projects associated with this domain. Mule applications can be associated with only one domain, but a domain can be associated with multiple projects. Shared resources allow multiple development teams to work in parallel using the same set of reusable connectors. Defining these connectors as shared resources at the domain level allows the team to: - Expose multiple services within the domain through the same port. - Share the connection to persistent storage. - Share services between apps through a well-defined interface. - Ensure consistency between apps upon any changes because the configuration is only set in one place.
* Use domains Project to share the same host and port among multiple projects. You can declare the http connector within a domain project and associate the domain project with other projects. Doing this also allows to control thread settings, keystore configurations, time outs for all the requests made within multiple applications. You may think that one can also achieve this by duplicating the http connector configuration across all the applications. But, doing this may pose a nightmare if you have to make a change and redeploy all the applications.
* If you use connector configuration in the domain and let all the applications use the new domain instead of a default domain, you will maintain only one copy of the http connector configuration. Any changes will require only the domain to the redeployed instead of all the applications.
You can start using domains in only three steps:
1) Create a Mule Domain project
2) Create the global connector configurations which needs to be shared across the applications inside the Mule Domain project
3) Modify the value of domain in mule-deploy.properties file of the applications



Use a certificate defined in already deployed Mule domain Configure the certificate in the domain so that the API proxy HTTPS Listener references it, and then deploy the secure API proxy to the target Runtime Fabric, or on-premises target. (CloudHub is not supported with this approach because it does not support Mule domains.)

Question: 3

An API client is implemented as a Mule application that includes an HTTP Request operation using a default configuration. The HTTP Request operation invokes an external API that follows standard HTTP status code conventions, which causes the HTTP Request operation to return a 4xx status code. What is a possible cause of this status code response?

A. An error occurred inside the external API implementation when processing the HTTP request that was received from the outbound HTTP Request operation of the Mule application
B. The external API reported that the API implementation has moved to a different external endpoint
C. The HTTP response cannot be interpreted by the HTTP Request operation of the Mule application after it was received from the external API
D. The external API reported an error with the HTTP request that was received from the outbound HTTP Request operation of the Mule application

Answer: D

E xpl anati on:

Correct choice is: "The external API reported an error with the HTTP request that was received from the outbound HTTP Request operation of the Mule application"
Understanding HTTP 4XX Client Error Response Codes : A 4XX Error is an error that arises in cases where there is a problem with the user's request, and not with the server.
Such cases usually arise when a user's access to a webpage is restricted, the user misspells the URL, or when a webpage is nonexistent or removed from the public's view.
In short, it is an error that occurs because of a mismatch between what a user is trying to access, and its availability to the user — either because the user does not have the right to access it, or because what the user is trying to access simply does not exist. Some of the examples of 4XX errors are
400 Bad Request The server could not understand the request due to invalid syntax. 401

Unauthorized Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response. 403 Forbidden The client does not have access rights to the content; that is, it is unauthorized, so the server is refusing to give the requested resource. Unlike 401, the client's identity is known to the server. 404 Not Found The server can not find the requested resource. In the browser, this means the URL is not recognized. In an API, this can also mean that the endpoint is valid but the resource itself does not exist. Servers may also send this response instead of 403 to hide the existence of a resource from an unauthorized client. This response code is probably the most famous one due to its frequent occurrence on the web. 405 Method Not Allowed The request method is known by the server but has been disabled and cannot be used. For example, an API may forbid DELETE-ing a resource. The two mandatory methods, GET and HEAD, must never be disabled and should not return this error code. 406 Not Acceptable This response is sent when the web server, after performing server-driven content negotiation, doesn't find any content that conforms to the criteria given by the user agent. The external API reported that the API implementation has moved to a different external endpoint cannot be the correct answer as in this situation 301 Moved Permanently The URL of the requested resource has been changed permanently. The new URL is given in the response. ---------- ---------- In Lay man's term the scenario would be: API CLIENT —> MuleSoft API - HTTP request “Hey, API.. process this” —> External API API CLIENT <– MuleSoft API - http response "I'm sorry Client.. something is wrong with that request" <– (4XX) External API

Question: 4

An XA transaction Is being configured that involves a JMS connector listening for Incoming JMS messages. What is the meaning of the timeout attribute of the XA transaction, and what happens after the timeout expires?

A. The time that is allowed to pass between committing the transaction and the completion of the Mule flow After the timeout, flow processing triggers an error
B. The time that Is allowed to pass between receiving JMS messages on the same JMS connection After the timeout, a new JMS connection Is established
C. The time that Is allowed to pass without the transaction being ended explicitly After the timeout, the transaction Is forcefully rolled-back
D. The time that Is allowed to pass for state JMS consumer threads to be destroyed After the timeout, a new JMS consumer thread is created

Answer: C

E xpl anati on:

* Setting a transaction timeout for the Bitronix transaction manager
  - Set the transaction timeout either
    – In wrapper.conf
    – In CloudHub in the Properties tab of the Mule application deployment
  - The default is 60 secs. It is defined as
    mule.bitronix.transactiontimeout = 120
* This property defines the timeout for each transaction created for this manager.
  If the transaction has not terminated before the timeout expires it will be automatically rolled

back.

Additional Info around Transaction Management:
Bitronix is available as the XA transaction manager for Mule applications

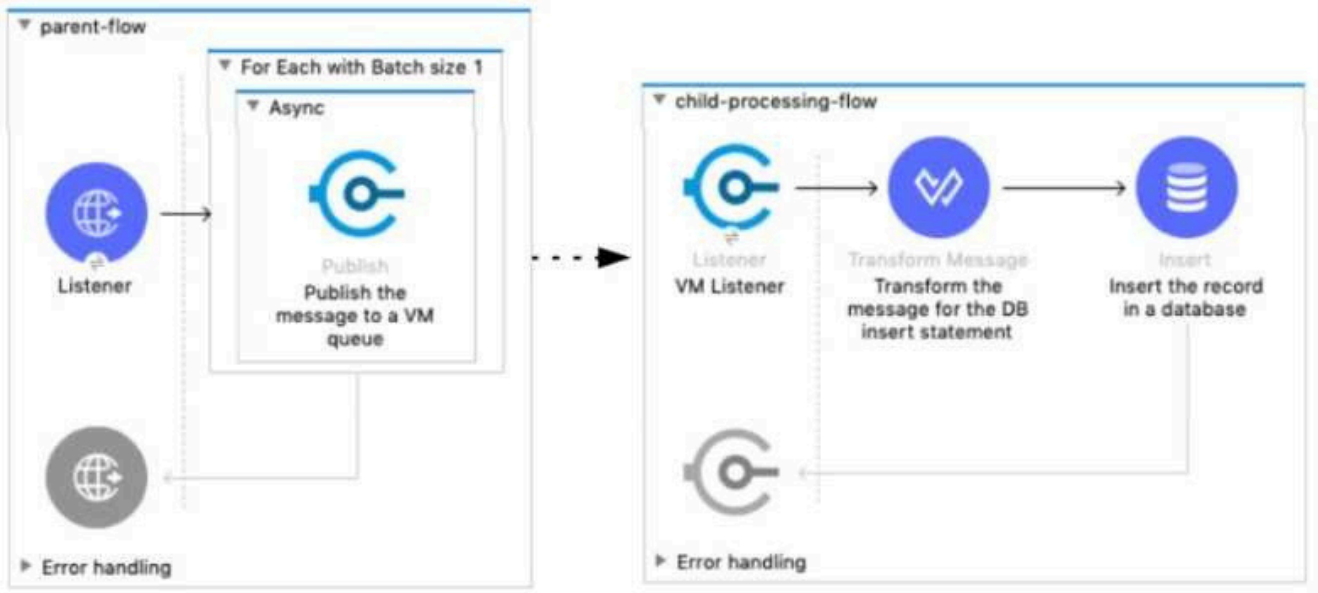
- To use Bitronix, declare it as a global configuration element in the Mule application
  <bti:transaction-manager />
- Each Mule runtime can have only one instance of a Bitronix transaction manager, which is shared by all Mule applications
- For customer-hosted deployments, define the XA transaction manager in a Mule domain
  – Then share this global element among all Mule applications in the Mule runtime

Transaction Management

Characteristics	Local Transactions	Extended Architecture (XA) Transactions
Connector Requisite 1	All operations inside the transaction must belong to same Connector.	Operations inside the transaction may belong to different Connectors
Connector Requisite 2	Connectors may not be XA enabled	Connectors must be XA enabled
Connector Requisite 3	Connectors should use single config reference	Connectors may use multiple config references
Available resources	JMS, VM, JDBC	JMS, VM, JDBC
Uses Two Phase Commit (2PC)	No	Yes
DB Operations	Perform database operation to only one database resource	Perform database operation to one or more transactional resource
Supports Nested Transactions	Does not support nested transactions.	Supports nested transactions.
Bitronix is available	No	Yes
A.C.I.D Properties	No	Yes
Performance	Better than XA	Latency Increases
Thread Pooling	BLOCKING_IO	BLOCKING_IO
Recovery is case of system failure	No	Using Bitronix

Question: 5

Refer to the exhibit.



A Mule 4 application has a parent flow that breaks up a JSON array payload into 200 separate items, then sends each item one at a time inside an Async scope to a VM queue.
A second flow to process orders has a VM Listener on the same VM queue. The rest of this flow processes each received item by writing the item to a database.
This Mule application is deployed to four CloudHub workers with persistent queues enabled.
What message processing guarantees are provided by the VM queue and the CloudHub workers, and how are VM messages routed among the CloudHub workers for each invocation of the parent flow under normal operating conditions where all the CloudHub workers remain online?

A. EACH item VM message is processed AT MOST ONCE by ONE CloudHub worker, with workers chosen in a deterministic round-robin fashion Each of the four CloudHub workers can be expected to process 1/4 of the Item VM messages (about 50 items)
B. EACH item VM message is processed AT LEAST ONCE by ONE ARBITRARY CloudHub worker Each of the four CloudHub workers can be expected to process some item VM messages
C. ALL Item VM messages are processed AT LEAST ONCE by the SAME CloudHub worker where the parent flow was invoked
This one CloudHub worker processes ALL 200 item VM messages
D. ALL item VM messages are processed AT MOST ONCE by ONE ARBITRARY CloudHub worker
This one CloudHub worker processes ALL 200 item VM messages

Answer: B

E xpl anati on:

Correct answer is EACH item VM message is processed AT LEAST ONCE by ONE ARBITRARY CloudHub worker. Each of the four CloudHub workers can be expected to process some item VM messages In Cloudhub, each persistent VM queue is listened on by every CloudHub worker - But each message is read and processed at least once by only one CloudHub worker and the duplicate processing is possible - If the CloudHub worker fails , the message can be read by another worker to prevent loss of messages and this can lead to duplicate processing - By default , every CloudHub worker's VM Listener receives different messages from VM Queue Referenece:
https://dzone.com/articles/deploying-mulesoft-application-on-1-worker-vs-mult