Free Questions for S90.08B by braindumpscollection

Shared by Ramsey on 24-05-2024

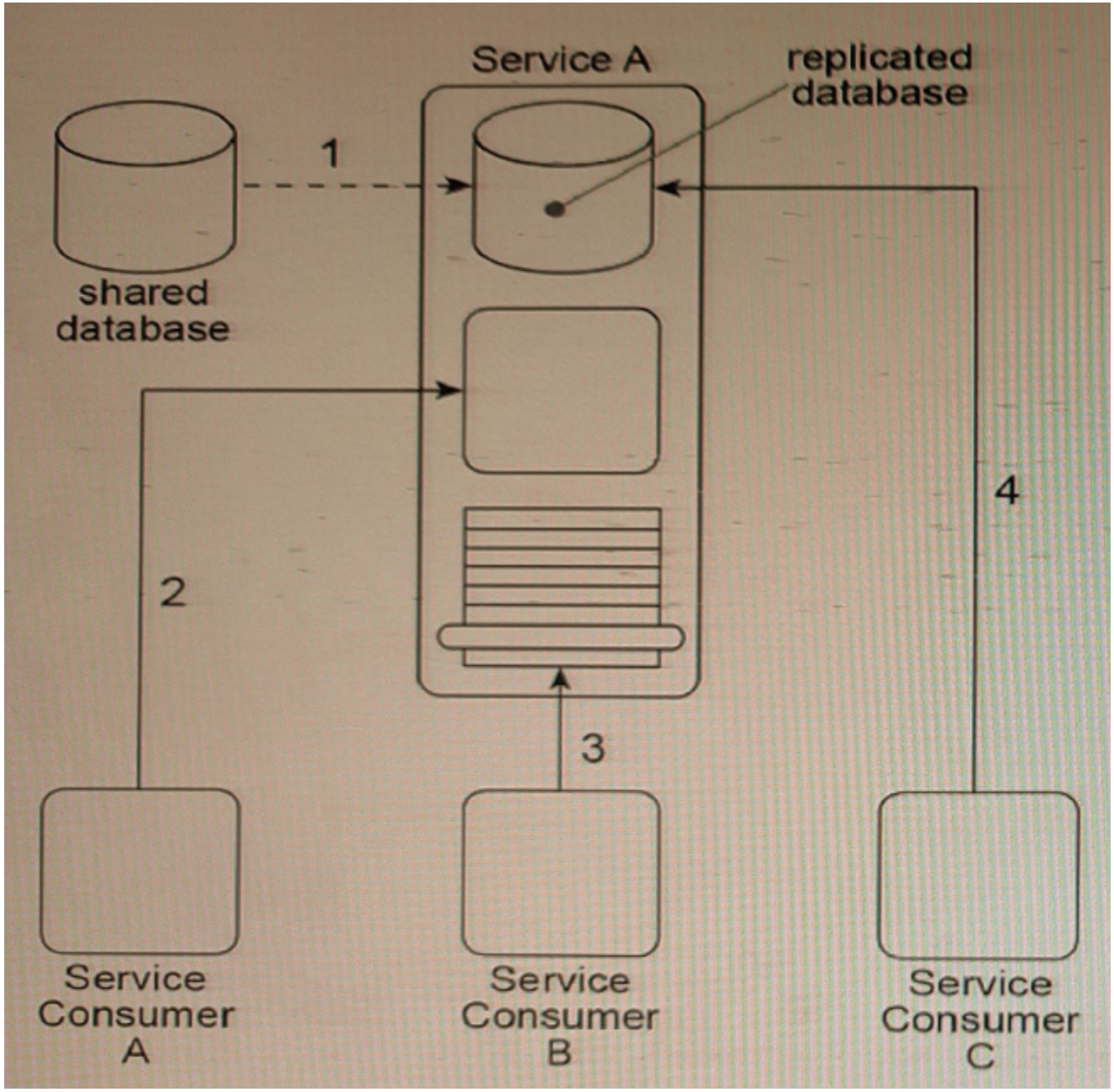
For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

Refer to Exhibit.


Service A
replicated database
1
shared database
2
3
4
Service Consumer A
Service Consumer B
Service Consumer C

Service A is a utility service that provides generic data access logic to a database containing data that is periodically replicated from a shared database (1). Because the Standardized Service Contract principle was applied to the design of Service A, its service contract has been fully standardized.

The service architecture of Service A Is being accessed by three service consumers. Service Consumer A accesses a component that is part of the Service A Implementation by Invoking it directly (2). Service Consumer B invokes Service A by accessing Its service contract (3). Service Consumer C directly accesses the replicated database that Is part of the Service A Implementation (4).

You've been told that the reason Service Consumers A and C bypass the published Service A service contract is because, for security reasons, they are not allowed to access a subset of the capabilities in the API that comprises the Service A service contract. How can the Service A architecture be changed to enforce these security restrictions while avoiding negative forms of coupling?

Options:

A- The Contract Centralization pattern can be applied to force all service consumers to access the Service A architecture via its published service contract. This will prevent negative forms of coupling that could lead to problems when the database is replaced. The Service Abstraction principle can then be applied to hide underlying service architecture details so that future service consumers cannot be designed to access any part of the underlying service implementation.

B- The Contract Centralization pattern can be applied to force service consumers to access the Service A architecture via its published service contract only. The Service Loose Coupling principle can then be applied to ensure that the centralized service contract does not contain any content that is dependent on or derived from the underlying service implementation.

C- The Contract Centralization pattern can be applied to force service consumers to access the Service A architecture via its published service contract only. The Concurrent Contracts pattern can be applied to Service A in order to establish one or more alternative service contracts. This allows service consumers with different levels of authorization to access different types of service logic via Service A's

published service contracts.

D- The Contract Centralization pattern can be applied to force service consumers to access the Service A architecture via its published service contract only. The Idempotent Capability pattern can be applied to Service A to establish alternative sets of service capabilities for service consumers with different levels of authorization.

Answer:

C

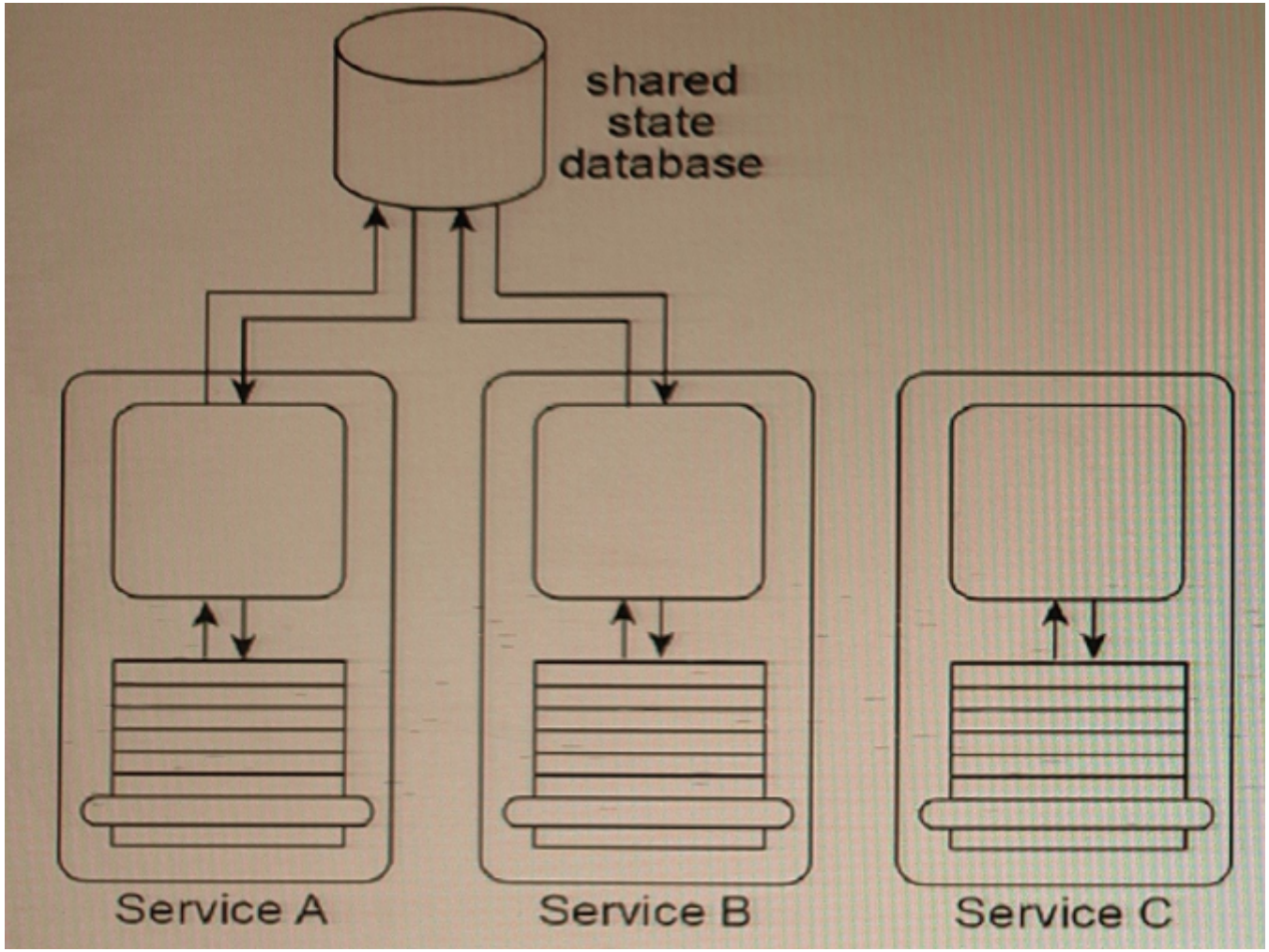
Explanation:

The Contract Centralization pattern can be applied to force service consumers to access the Service A architecture via its published service contract only. The Service Loose Coupling principle can then be applied to ensure that the centralized service contract does not contain any content that is dependent on or derived from the underlying service implementation. This will enforce the security restrictions while avoiding negative forms of coupling. By ensuring loose coupling, changes to the implementation of Service A will not require changes to its published service contract, making it easier to maintain and evolve the service.

Question 2

Question Type: MultipleChoice

Refer to Exhibit.



Services A, B, and C are non-agnostic task services. Service A and Service B use the same shared state database to defer their state data at runtime.

An assessment of the three services reveals that each contains some agnostic logic that cannot be made available for reuse because it is bundled together with non-agnostic logic.

The assessment also determines that because Service A, Service B and the shared state database are each located in physically separate environments, the remote communication required for Service A and Service B to interact with the shared state database is causing an unreasonable decrease in runtime performance.

How can the application of the Orchestration pattern improve this architecture?

Options:

A- The application of the Orchestration pattern will result in an environment whereby the Official Endpoint, State Repository, and Service Data Replication patterns are automatically applied, allowing the shared state database to be replicated via official service endpoints for Services A and B so that each task service can have its own dedicated state database.

B- The application of the Orchestration pattern will result in an environment whereby the non-agnostic logic can be cleanly separated from the agnostic logic that exists in Services A, B, and C, resulting in the need to design new agnostic services with reuse potential assured through the application of the Service Reusability principle. The State Repository pattern, which is supported by and local to the orchestration environment, provides a central state database that can be shared by Services A and B. The local state database avoids problems with remote communication.

C- The application of the Orchestration pattern will result in an environment whereby the Compensating Service Transaction is automatically applied, resulting In the opportunity to create sophisticated exception logic that can be used to compensate for the performance problems caused by Services A and B having to remotely access the state database. The API Gateway and Service Broker patterns are also automatically applied, providing common transformation functions in a centralized processing layer to help overcome any disparity in the service contracts that will need to be created for the new agnostic services.

D- The Orchestration pattern is not applicable to this architecture because it does not support the hosting of the required state repository.

Answer:

B

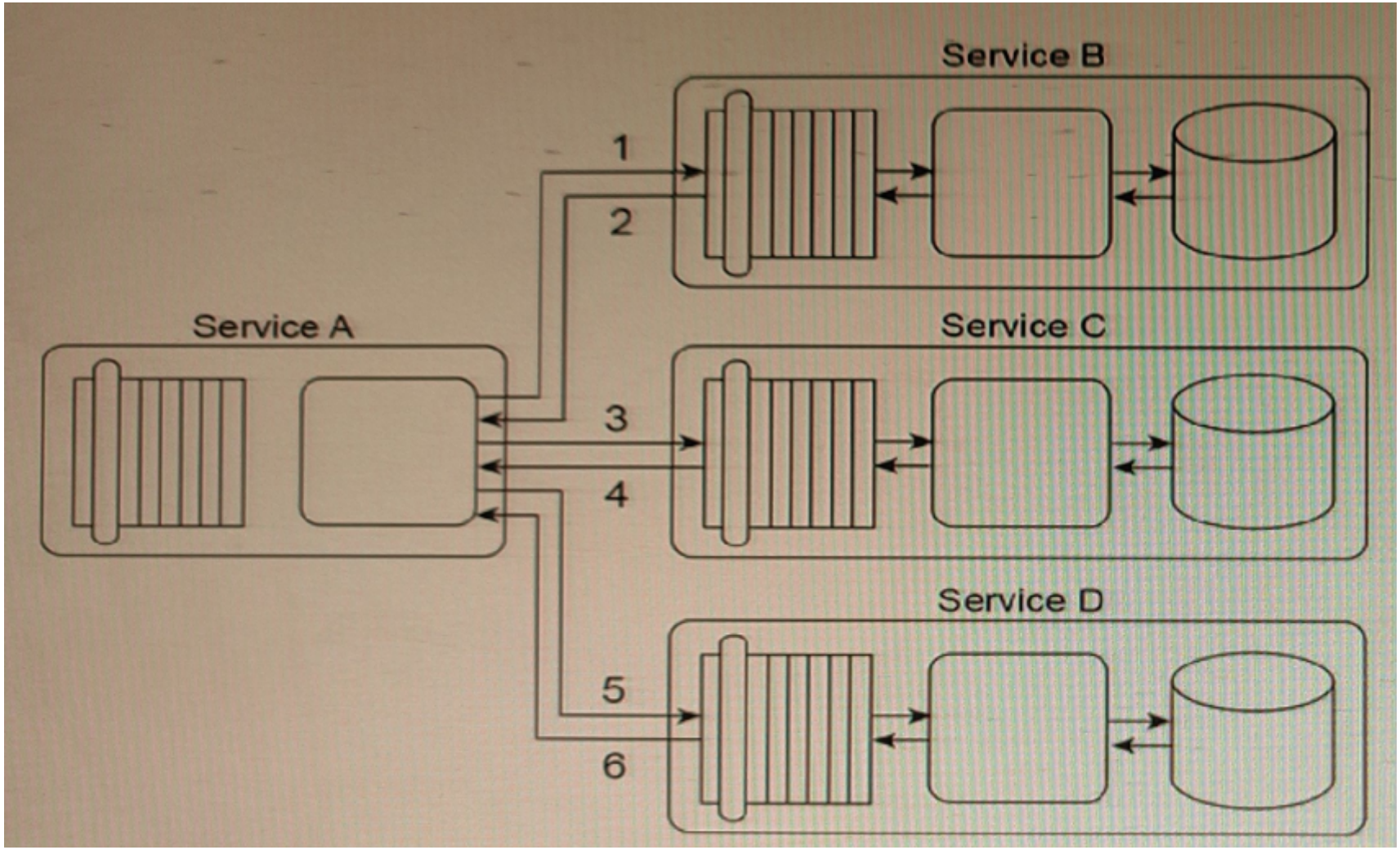
Explanation:

The application of the Orchestration pattern can improve this architecture by cleanly separating the non-agnostic logic from the agnostic logic, allowing the design of new agnostic services with reuse potential. The State Repository pattern, which is supported by and local to the orchestration environment, provides a central state database that can be shared by Services A and B. The local state database avoids problems with remote communication. Additionally, the Orchestration pattern provides a central controller that can coordinate the interactions between Services A, B, and C, reducing the need for remote communication between services and improving runtime performance.

Question 3

Question Type: MultipleChoice

Refer to Exhibit.



Service A is a task service that is required to carry out a series of updates to a set of databases in order to complete a task. To perform the database updates. Service A must interact with three other services that each provides standardized data access capabilities.

Service A sends its first update request message to Service B (1), which then responds with a message containing either a success or failure code (2). Service A then sends its second update request message to Service C (3), which also responds with a message containing either a success or failure code (4). Finally, Service A sends a request message to Service D (5), which responds with its own message containing either a success or failure code (6).

Services B, C and D are agnostic services that are reused and shared by multiple service consumers. This has caused unacceptable performance degradation for the service consumers of Service A as it is taking too long to complete its overall task. You've been asked to enhance the service composition architecture so that Service A provides consistent and predictable runtime performance. You are furthermore notified that a new type of data will be introduced to all three databases. It is important that this data is exchanged in a standardized manner so that the data model used for the data in inter-service messages is the same.

What steps can be taken to fulfill these requirements?

Options:

A- The Compensating Service Transaction pattern can be applied so that exception logic is executed to notify Service A whenever the data access logic executed by Service B, C, or D takes too long. If the execution time exceeds a predefined limit, then the overall service activity is cancelled and a failure code is returned to Service A. The Schema Centralization pattern is applied to ensure that all services involved in the composition use the same schemas to represented the data consistently.

B- The Composition Autonomy pattern can be applied to establish an isolated environment in which redundant implementations of Services B, C and D are accessed only by Service A. The Canonical Schema pattern can be applied to ensure that the new type of data is represented by the same data model, regardless of which service sends or receives a message containing the data.

C- The Redundant Implementation pattern is applied to Service A, along with the Service Instance Routing pattern. This allows for multiple instances of Service A to be created across multiple physical implementations, thereby increasing scalability and availability.

The Dual Protocols pattern is applied to all services to support proprietary and standardized data models.

D- The Service Fagade pattern is applied to all services in order to create an intermediary processing layer within each service architecture. The Content Negotiation pattern is applied so that each service fagade component within each service architecture is equipped with the logic required to defer request messages to other service instances when concurrent usage of the service is high, and to further apply the conversation logic necessary to convert proprietary data from a database into the standardized XML schema format.

Answer:

B

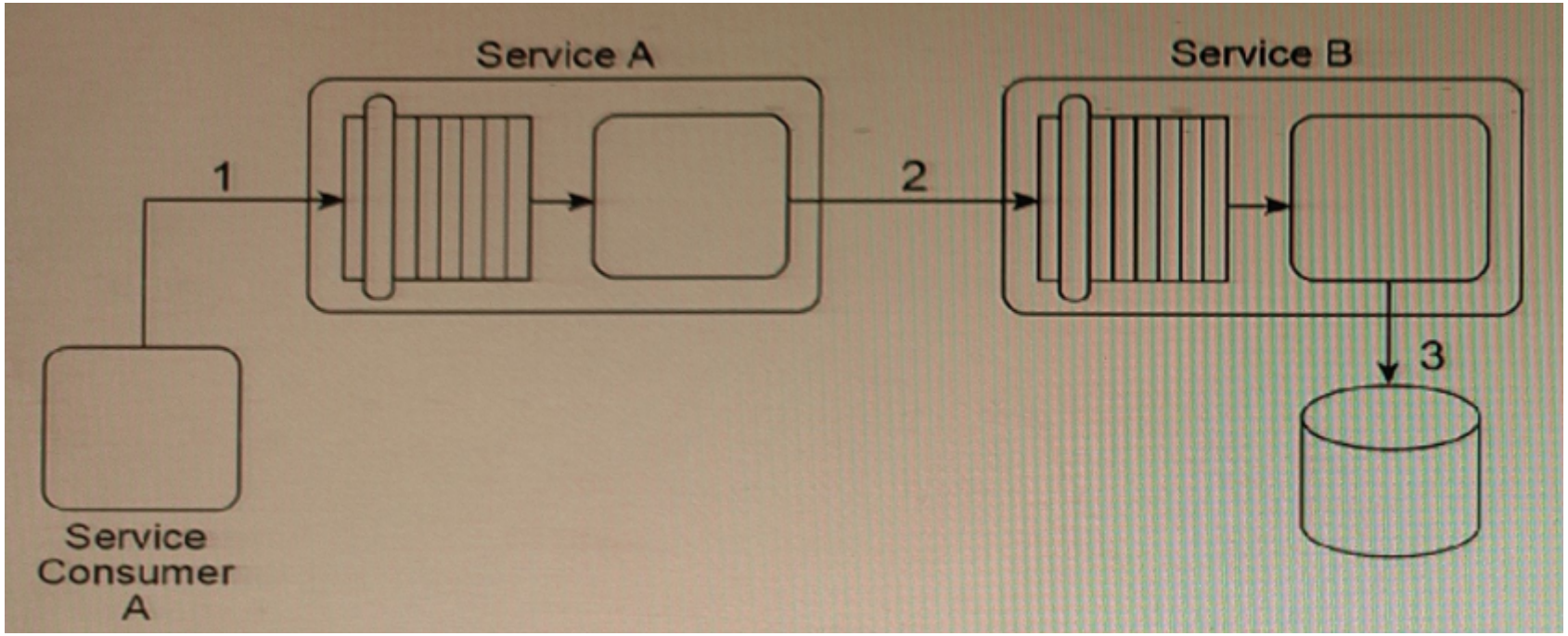
Explanation:

This approach isolates the services used by Service A, allowing it to avoid the performance degradation caused by multiple service consumers. By creating redundant implementations of Services B, C, and D that are accessed only by Service A, the Composition Autonomy pattern also ensures that Service A's runtime performance is consistent and predictable. Applying the Canonical Schema pattern ensures that the new type of data is exchanged in a standardized manner, ensuring consistent representation of the data model used for the data in inter-service messages.

Question 4

Question Type: MultipleChoice

Refer to Exhibit.



Service A is a SOAP-based Web service with a functional context dedicated to invoice-related processing. Service B is a REST-based utility service that provides generic data access to a database.

In this service composition architecture, Service Consumer A sends a SOAP message containing an invoice XML document to Service A (1). Service A then sends the invoice XML document to Service B (2), which then writes the invoice document to a database (3).

The data model used by Service Consumer A to represent the invoice document is based on XML Schema

Options:

A- The service contract of Service A is designed to accept invoice documents based on XML Schema B. The service contract for Service B is designed to accept invoice documents based on XML Schema A. The database to which Service B needs to write the invoice record only accepts entire business documents in a proprietary Comma Separated Value (CSV) format.
Due to the incompatibility of the XML schemas used by the services, the sending of the invoice document from Service Consumer A through to Service B cannot be accomplished using the services as they currently exist. Assuming that the Contract Centralization pattern is being applied and that the Logic Centralization pattern is not being applied, what steps can be taken to enable the sending of the invoice document from Service Consumer A to the database without adding logic that will increase the runtime performance requirements?

A- Service Consumer A can be redesigned to use XML Schema B so that the SOAP message it sends is compliant with the service contract of Service A. The Data Model Transformation pattern can then be applied to transform the SOAP message sent by Service A so that it conforms to the XML Schema A used by Service B. The Standardized Service Contract principle must then be applied to Service B and Service Consumer A so that the invoice XML document is optimized to avoid unnecessary validation.

B- The service composition can be redesigned so that Service Consumer A sends the invoice document directly to Service B after the specialized invoice processing logic from Service A is copied to Service B. Because Service Consumer A and Service B use XML Schema A, the need for transformation logic is avoided. This naturally applies the Service Loose Coupling principle because Service Consumer A is not required to send the invoice document In a format that is compliant with the database used by Service B.

C- Service Consumer A can be redesigned to write the invoice document directly to the database. This reduces performance requirements by avoiding the involvement of Service A and Service B. It further supports the application of the Service Loose Coupling principle by ensuring that Service Consumer A contains data access logic that couples it directly to the database.

D- The service composition can be redesigned so that Service Consumer A sends the invoice document directly to Service B. Because Service Consumer A and Service B use XML Schema A, the need for transformation logic is avoided. This naturally applies the Logic Centralization pattern because Service Consumer A is not required to send the invoice document In a format that is compliant with the

database used by Service B.

Answer:

A, A

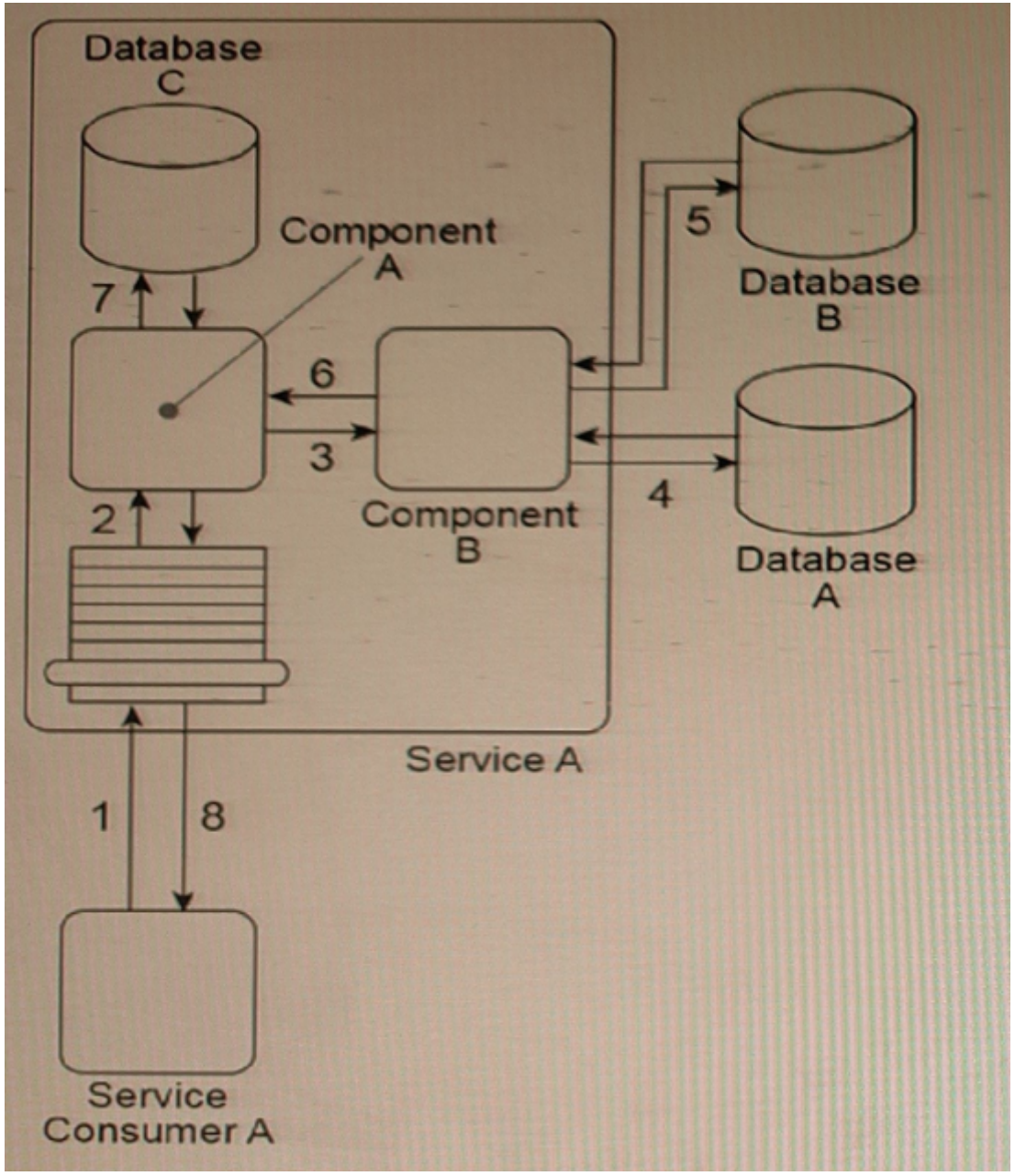
Explanation:

The recommended solution is to use the Data Model Transformation pattern to transform the invoice XML document from Schema B to Schema A before passing it to Service B. This solution maintains the separation of concerns and allows each service to work with its own specific XML schema. Additionally, the Standardized Service Contract principle should be applied to Service B and Service Consumer A to ensure that unnecessary validation is avoided, thus optimizing the invoice XML document. This solution avoids adding logic that will increase the runtime performance requirements.

Question 5

Question Type: MultipleChoice

Refer to Exhibit.


Database
C
Component
A
7
6
3
2
Component
B
5
Database
B
4
Database
A
Service A
1
8
Service
Consumer A

Service Consumer A sends Service A a message containing a business document (1). The business document is received by Component A, which keeps the business document in memory and forwards a copy to Component B (3). Component B first writes portions of the business document to Database A (4). Component B then writes the entire business document to Database B and uses some of the data values from the business document as query parameters to retrieve new data from Database B (5).

Next, Component B returns the new date* back to Component A (6), which merges it together with the original business document it has been keeping in memory and then writes the combined data to Database C (7). The Service A service capability invoked by Service Consumer A requires a synchronous request-response data exchange. Therefore, based on the outcome of the last database update, Service A returns a message with a success or failure code back to Service Consumer A (8).

Databases A and B are shared, and Database C is dedicated to the Service A service architecture.

There are several problems with this architecture. The business document that Component A is required to keep in memory (while it waits for Component B to complete its processing) can be very large. The amount of runtime resources Service A uses to keep this data in memory can decrease the overall performance of all service instances, especially when it is concurrently invoked by multiple service consumers. Additionally, Service A can take a long time to respond back to Service Consumer A because Database A is a shared database that sometimes takes a long time to respond to Component B. Currently, Service Consumer A will wait for up to 30 seconds for a response, after which it will assume the request to Service A has failed and any subsequent response messages from Service A will be rejected.

What steps can be taken to solve these problems?

Options:

A- The Service Statelessness principle can be applied together with the State Repository pattern to extend Database C so that it also becomes a state database allowing Component A to temporarily defer the business document data while it waits for a response from Component B. The Service Autonomy principle can be applied together with the Legacy Wrapper pattern to isolate Database A so that it is encapsulated by a separate wrapper utility service. The Compensating Service Transaction pattern can be applied so that whenever Service A's response time exceeds 30 seconds, a notification is sent to a human administrator to raise awareness of the fact that the eventual response of Service A will be rejected by Service Consumer A.

B- The Service Statelessness principle can be applied together with the State Repository pattern to establish a state database to which Component A can defer the business document data to while it waits for a response from Component B. The Service Autonomy principle can be applied together with the Service Data Replication pattern to establish a dedicated replicated database for Component B to access instead of shared Database A. The Asynchronous Queuing pattern can be applied to establish a message queue between Service Consumer A and Service A so that Service Consumer A does not need to remain stateful while it waits for a response from Service A.

C- The Service Statelessness principle can be applied together with the State Repository pattern to establish a state database to which Component A can defer the business document data while it waits for a response from Component B. The Service Autonomy principle can be applied together with the Service Abstraction principle, the Legacy Wrapper pattern, and the Service Fagade pattern in order to isolate Database A so that it is encapsulated by a separate wrapper utility service and to hide the Database A implementation from Service A and to position a fagade component between Component B and the new wrapper service. This fagade component will be responsible for compensating the unpredictable behavior of Database A.

D- None of the above.

Answer:

B

Explanation:

The problems with the current architecture can be addressed by applying the following patterns:

Service Statelessness principle and State Repository pattern - This pattern allows Component A to defer the business document data to a state database while it waits for a response from Component B. This helps reduce the amount of runtime resources Service A uses to keep the data in memory and improves overall performance.

Service Autonomy principle and Service Data Replication pattern - This pattern allows Component B to access a dedicated replicated database instead of the shared Database A, which can improve response time.

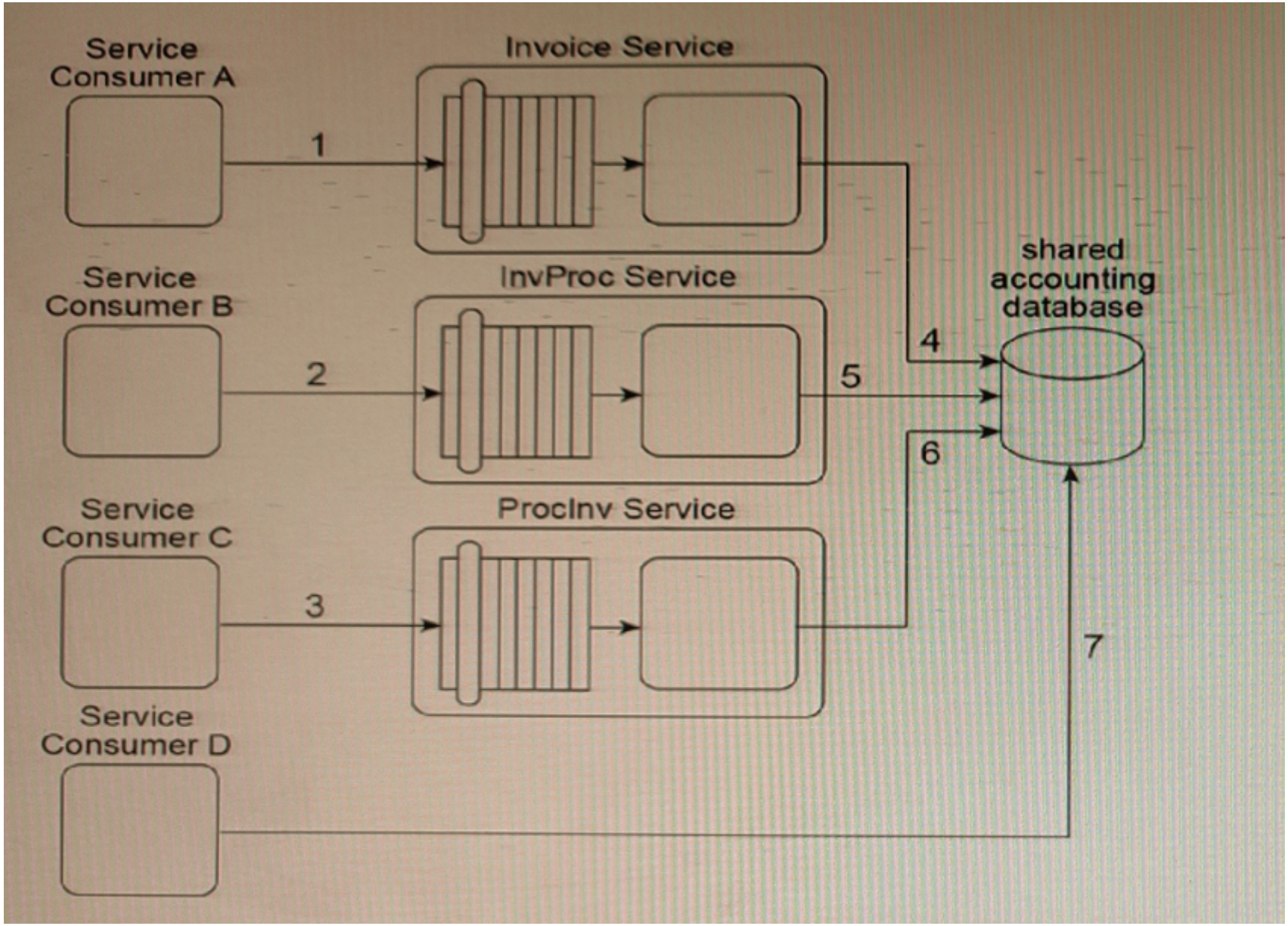
Asynchronous Queuing pattern - This pattern allows Service A to use a message queue to communicate with Service Consumer A asynchronously. This means that Service Consumer A does not need to remain stateful while waiting for a response from Service A, which can improve overall performance and scalability.

Therefore, option B is the correct answer. Option A is incorrect because it suggests using the Compensating Service Transaction pattern to raise awareness of the eventual response rejection, which does not actually solve the problem. Option C is also incorrect because it suggests using multiple patterns, which may not be necessary and can add unnecessary complexity to the architecture.

Question 6

Question Type: MultipleChoice

Refer to Exhibit.


Service Consumer A
Invoice Service
1
Service Consumer B
InvProc Service
2
shared accounting database
4
5
6
Service Consumer C
ProcInv Service
3
7
Service Consumer D

Our service inventory contains the following three services that provide Invoice-related data access capabilities: Invoice, InvProc and Proclnv. These services were created at different times by different project teams and were not required to comply with any design standards. Therefore, each of these services has a different data model for representing invoice data.

Currently, each of these three services has a different service consumer: Service Consumer A accesses the Invoice service (1), Service Consumer B (2) accesses the InvProc service, and Service Consumer C (3) accesses the Proclnv service. Each service consumer invokes a data access capability of an invoice-related service, requiring that service to interact with the shared accounting database that is used by all invoice-related services (4, 5, 6).

Additionally, Service Consumer D was designed to access invoice data from the shared accounting database directly (7). (Within the context of this architecture, Service Consumer D is labeled as a service consumer because it is accessing a resource that is related to the illustrated service architectures.)

Assuming that the Invoice service, InvProc service and Proclnv service are part of the same service inventory, what steps would be required to fully apply the Official Endpoint pattern?

Options:

A- One of the invoice-related services needs to be chosen as the official service providing invoice data access capabilities. Service Consumers A, B, and C then need to be redesigned to only access the chosen invoice-related service. Because Service Consumer D does not rely on an invoice-related service, it is not affected by the Official Endpoint pattern and can continue to access the accounting database directly. The Service Abstraction principle can be further applied to hide the existence of the shared accounting database and other implementation details from current and future service consumers.

B- One of the invoice-related services needs to be chosen as the official service providing invoice data access capabilities and logic from the other two services needs to be moved to execute within the context of the official Invoice service. Service Consumers A, B, and C

then need to be redesigned to only access the chosen invoice-related service. Service Consumer D also needs to be redesigned to not access the shared accounting database directly, but to also perform its data access by interacting with the official invoice-related service. The Service Abstraction principle can be further applied to hide the existence of the shared accounting database and other implementation details from current and future service consumers.

C- Because Service Consumers A, B, and C are already carrying out their data access via published contracts, they are not affected by the Official Endpoint pattern. Service Consumer D needs to be redesigned so that it does not access the shared accounting database directly, but instead performs its data access by interacting with the official invoice-related service. The Service Abstraction principle can be further applied to hide the existence of the shared accounting database and other implementation details from current and future service consumers.

D- One of the invoice-related services needs to be chosen as the official service providing invoice data access capabilities. Because Service Consumer D does not rely on an invoice-related service, it is not affected by the Official Endpoint pattern and can continue to access the accounting database directly. The Service Loose Coupling principle can be further applied to decouple Service Consumers A, B, and C from the shared accounting database and other implementation details.

Answer:

B

Explanation:

he Legacy Wrapper pattern can be applied so that Component B is separated into a separate utility service that wraps the shared database. The Legacy Wrapper pattern can be applied again so that Component C is separated into a separate utility service that acts as a wrapper for the legacy system API. The Legacy Wrapper pattern can be applied once more to Component D so that it is separated into another utility service that provides standardized access to the file folder. The Service Facade pattern can be applied so that three

facade components are added: one between Component A and each of the new wrapper utility services. This way, the facade components can compensate for any change in behavior that may occur as a result of the separation. The Service Composability principle can be further applied to Service A and the three new wrapper utility services so that all four services are optimized for participation in the new service composition. This will help make up for any performance loss that may result from splitting the three components into separate services.

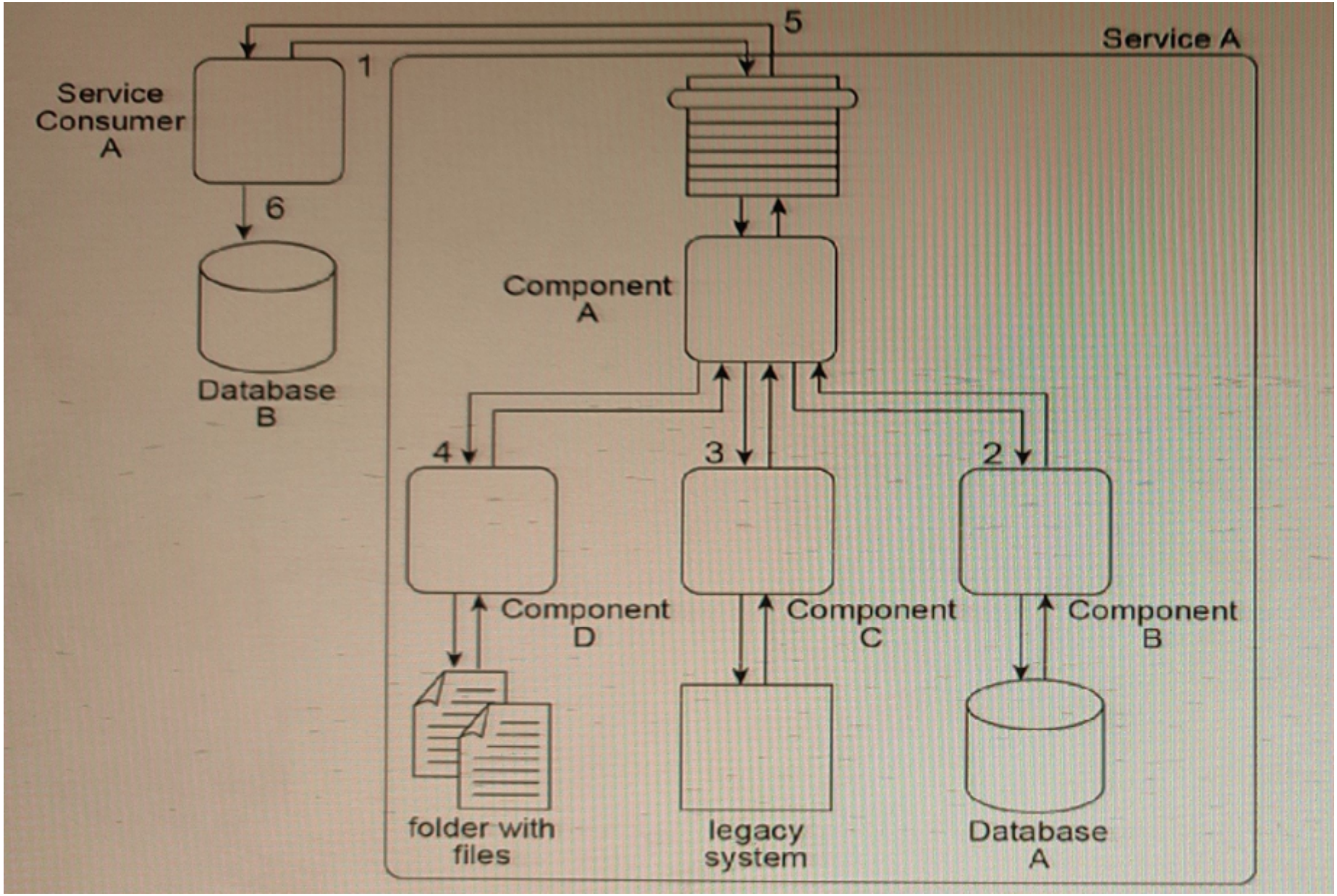
By applying the Legacy Wrapper pattern to separate Components B, C, and D into three different utility services, the shared resources within the IT enterprise (Database A, the legacy system, and the file folders) can be properly encapsulated and managed by dedicated services. The Service Facade pattern can then be used to create a facade component between Component A and each of the new wrapper utility services, allowing them to interact seamlessly without affecting Service Consumer A's behavior.

Finally, the Service Composability principle can be applied to ensure that Service A and the three new wrapper utility services are optimized for participation in the new service composition. This will help to mitigate any performance loss that may result from splitting the three components into separate services.

Question 7

Question Type: **MultipleChoice**

Refer to Exhibit.


Service A
Service Consumer A
1
5
6
Database B
Component A
4
3
2
Component D
Component C
Component B
folder with files
legacy system
Database A

When Service A receives a message from Service Consumer A (1), the message is processed by Component

Options:

A- This component first invokes Component B (2), which uses values from the message to query Database A in order to retrieve additional data. Component B then returns the additional data to Component A. Component A then invokes Component C (3), which interacts with the API of a legacy system to retrieve a new data value. Component C then returns the data value back to Component A. Next, Component A sends some of the data It has accumulated to Component D (4), which writes the data to a text file that is placed in a specific folder. Component D then waits until this file is imported into a different system via a regularly scheduled batch import. Upon completion of the import, Component D returns a success or failure code back to Component A. Component A finally sends a response to Service Consumer A (5) containing all of the data collected so far and Service Consumer A writes all of the data to Database B (6). Components A, B, C, and D belong to the Service A service architecture. Database A, the legacy system and the file folders are shared resources within the IT enterprise.
Service A is an entity service with a service architecture that has grown over the past few years. As a result of a service inventory-wide redesign project, you are asked to revisit the Service A service architecture in order to separate the logic provided by Components B, C, and D into three different utility services without disrupting the behavior of Service A as it relates to Service Consumer A.
What steps can be taken to fulfill these requirements?

A- The Legacy Wrapper pattern can be applied so that Component B is separated into a separate wrapper utility service that wraps the shared database. The Asynchronous Queuing pattern can be applied so that a messaging queue is positioned between Component A and Component C, thereby enabling communication during the times when the legacy system may be unavailable or heavily accessed by other parts of the IT enterprise. The Service Fagade pattern can be applied so that a fagade component is added between Component A and Component D so that any change In behavior can be compensated. The Service Autonomy principle can be further applied to Service A to help make up for any performance loss that may result from splitting the component into a separate wrapper utility service.

B- The Legacy Wrapper pattern can be applied so that Component B Is separated into a separate utility service that wraps the shared database. The Legacy Wrapper pattern can be applied again so that Component C is separated into a separate utility service that acts as a wrapper for the legacy system API. The Legacy Wrapper pattern can be applied once more to Component D so that it is separated into another utility service that provides standardized access to the file folder. The Service Fagade pattern can be applied so that three fagade components are added: one between Component A and each of the new wrapper utility services. This way, the fagade components can compensate for any change in behavior that may occur as a result of the separation. The Service Composability principle can be further applied to Service A and the three new wrapper utility services so that all four services are optimized for participation in the new service composition. This will help make up for any performance loss that may result from splitting the three components into separate services.

C- The Legacy Wrapper pattern can be applied so that Component B is separated into a separate utility service that wraps the shared database. The Legacy Wrapper pattern can be applied again so that Component C is separated into a separate utility service that acts as a wrapper for the legacy system API. Component D can also be separated into a separate service and the Event-Driven Messaging pattern can be applied to establish a publisher-subscriber relationship between this new service and Component A. The interaction between Service Consumer A and Component A can then be redesigned so that Component A first interacts with Component B and the new wrapper service. Service A then issues a final message back to Service Consumer A. The Service Composability principle can be further applied to Service A and the three new wrapper utility services so that all four services are optimized for participation in the new service composition. This will help make up for any performance loss that may result from splitting the three components into separate services.

D- The Legacy Wrapper pattern can be applied so that Component B is separated into a separate wrapper utility service that wraps the shared database. The State Repository and State Messaging patterns can be applied so that a messaging repository is positioned between Component A and Component C, thereby enabling meta data-driven communication during the times when the legacy system may be unavailable or heavily accessed by other parts of the IT enterprise. The Service Fagade pattern can be applied so that a fagade component is added between Component A and Component D so that any change in behavior can be compensated. The Service Statelessness principle can be further applied to Service A to help make up for any performance loss that may result from splitting the component into a separate wrapper utility service.

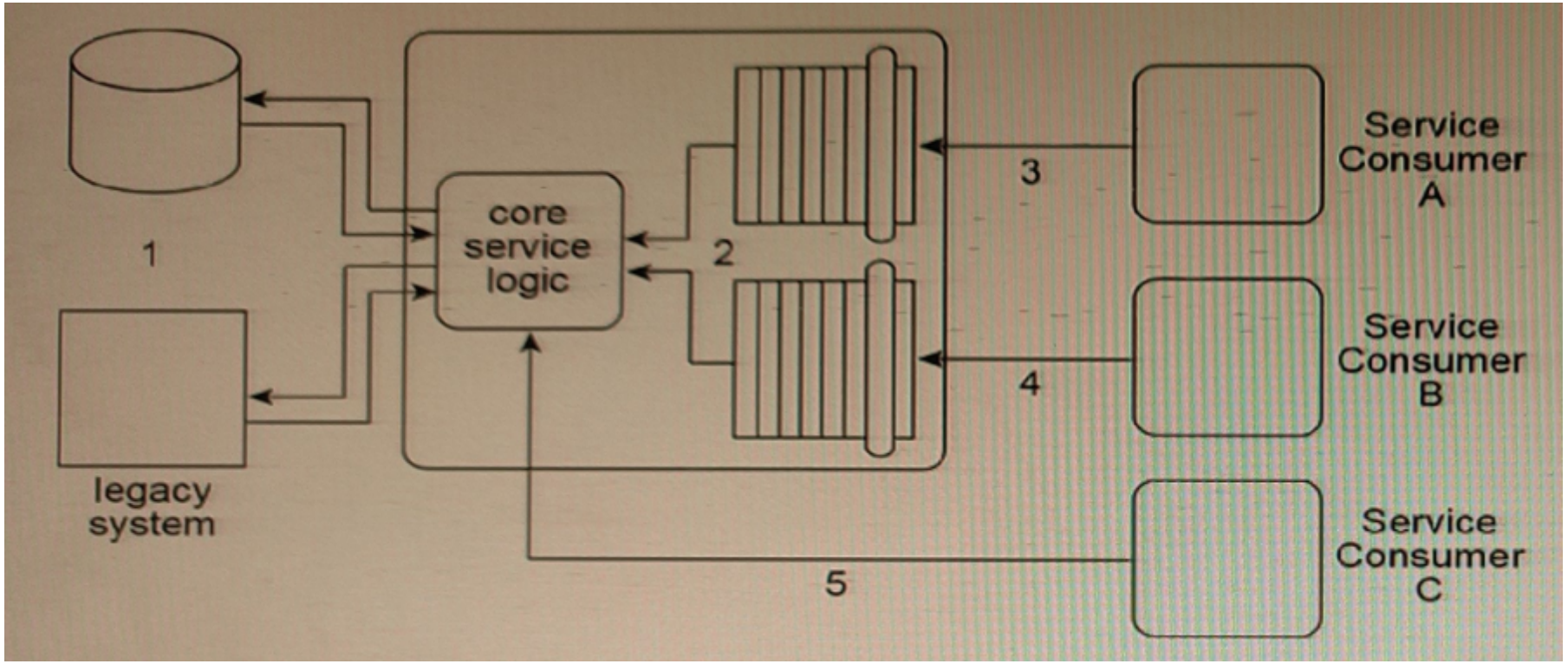
Answer:

B

Question 8

Question Type: MultipleChoice

Refer to Exhibit.



The architecture for Service A displayed in the figure shows how the core logic of Service A has expanded over time to connect to a database and a proprietary legacy system (1), and to support two separate service contracts (2) that are accessed by different service consumers.

The service contracts are fully decoupled from the service logic. The service logic is therefore coupled to the service contracts and to the underlying implementation resources (the database and the legacy system).

Service A currently has three service consumers. Service Consumer A and Service Consumer B access Service A's two service contracts (3, 4). Service Consumer C bypasses the service contracts and accesses the service logic directly (5).

You are told that the database and legacy system that are currently being used by Service A are being replaced with different products. The two service contracts are completely decoupled from the core service logic, but there is still a concern that the introduction of the new products will cause the core service logic to behave differently than before.

What steps can be taken to change the Service A architecture in preparation for the introduction of the new products so that the impact on Service Consumers A and B is minimized? What further step can be taken to avoid consumer-to-implementation coupling with Service Consumer C?

Options:

A- The Service Fagade pattern can be applied to position fagade components between the core service logic and Service Consumers A and B. These fagade components will be designed to regulate the behavior of Service A. The Service Abstraction principle can be applied to hide the implementation details of the core service logic of Service A, thereby shielding this logic from changes to the implementation. The Schema Centralization pattern can be applied to force Service Consumer C to access Service A via one of its existing service contracts.

B- A third service contract can be added together with the application of the Contract Centralization pattern. This will force Service Consumer C to access Service A via the new service contract. The Service Fagade pattern can be applied to position a fagade component between the new service contract and Service Consumer C in order to regulate the behavior of Service A. The Service Abstraction principle can be applied to hide the implementation details of Service A so that no future service consumers are designed to access any of Service A's underlying resources directly.

C- The Service Fagade pattern can be applied to position fagade components between the core service logic and the two service contracts. These fagade components will be designed to regulate the behavior of Service A. The Service Loose Coupling principle can be applied to avoid negative forms of coupling.

D- The Service Fagade pattern can be applied to position fagade components between the core service logic and the implementation resources (the database and the legacy system). These fagade components will be designed to insulate the core service logic of Service A from the changes in the underlying implementation resources. The Schema Centralization and Endpoint Redirection patterns can also be applied to force Service Consumer C to access Service A via one of its existing service contracts.

Answer:

D

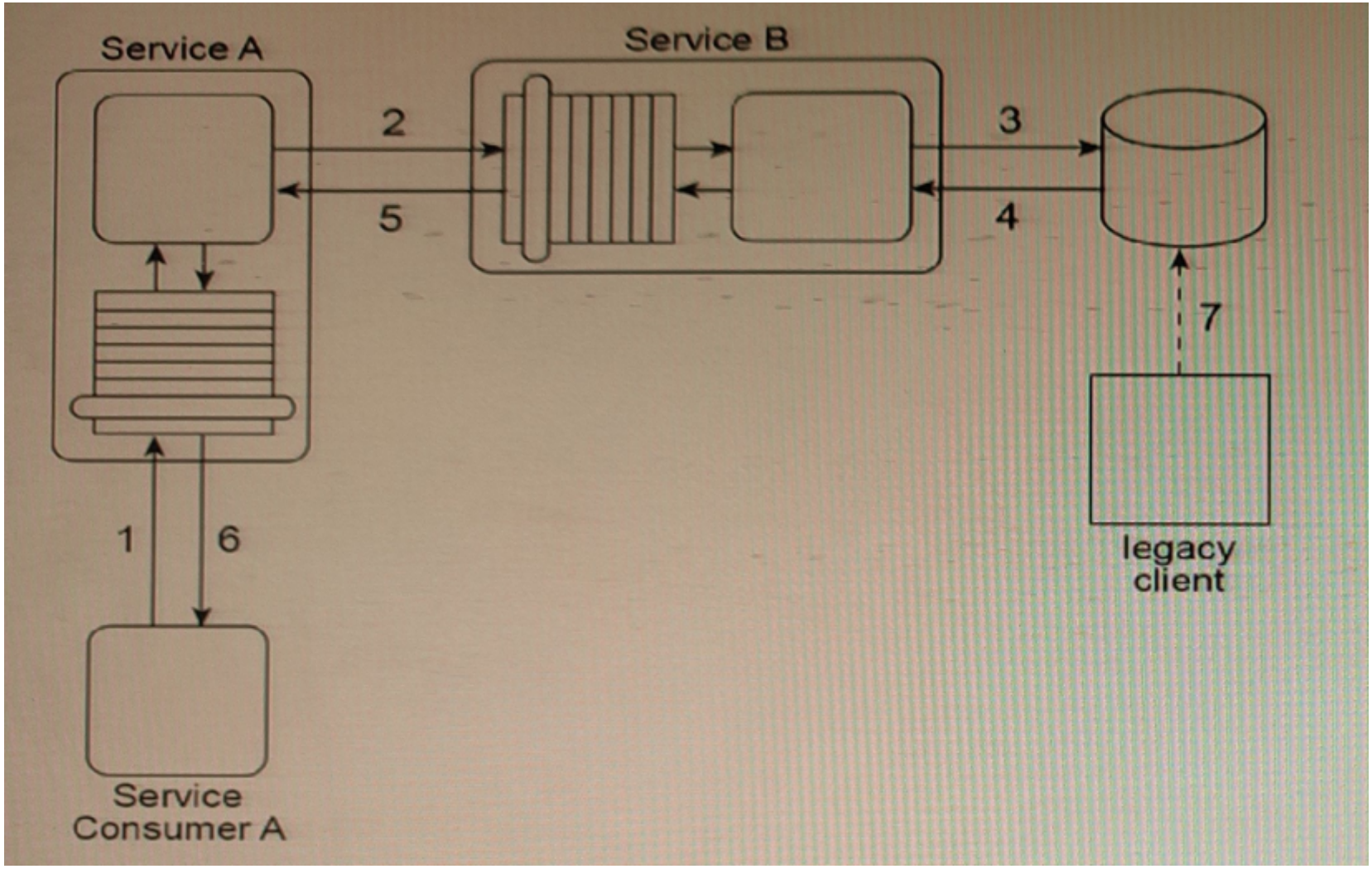
Explanation:

The Service Fagade pattern can be applied to position fagade components between the core service logic and the implementation resources (the database and the legacy system). These fagade components will be designed to insulate the core service logic of Service A from the changes in the underlying implementation resources. This will minimize the impact of the introduction of the new products on Service Consumers A and B since the service contracts are fully decoupled from the core service logic. The Schema Centralization and Endpoint Redirection patterns can also be applied to force Service Consumer C to access Service A via one of its existing service contracts, avoiding direct access to the core service logic and the underlying implementation resources.

Question 9

Question Type: MultipleChoice

Refer to Exhibit.



Service A is an entity service that provides a Get capability which returns a data value that is frequently changed.

Service Consumer A invokes Service A in order to request this data value (1). For Service A to carry out this request, it must invoke Service B (2), a utility service that interacts (3, 4) with the database in which the data value is stored. Regardless of whether the data value changed, Service B returns the latest value to Service A (5), and Service A returns the latest value to Service Consumer A (6).

The data value is changed when the legacy client program updates the database (7). When this change will occur is not predictable. Note also that Service A and Service B are not always available at the same time.

Any time the data value changes, Service Consumer A needs to receive It as soon as possible. Therefore, Service Consumer A initiates the message exchange shown In the figure several times a day. When it receives the same data value as before, the response from Service A Is ignored. When Service A provides an updated data value, Service Consumer A can process it to carry out its task.

The current service composition architecture is using up too many resources due to the repeated invocation of Service A by Service Consumer A and the resulting message exchanges that occur with each invocation.

What steps can be taken to solve this problem?

Options:

A- The Event-Driven Messaging pattern can be applied by establishing a subscriber-publisher relationship between Service A and Service B. This way, every time the data value is updated, an event is triggered and Service B, acting as the publisher, can notify Service A, which acts as the subscriber. The Asynchronous Queuing pattern can be applied between Service A and Service B so that the event notification message sent out by Service B will be received by Service A, even when Service A is unavailable.

B- The Event-Driven Messaging pattern can be applied by establishing a subscriber-publisher relationship

between Service Consumer A and Service A. This way, every time the data value is updated, an event is triggered and Service A, acting as the publisher, can notify Service Consumer A, which acts as the subscriber. The Asynchronous Queuing pattern can be applied between Service Consumer A and Service A so that the event notification message sent out by Service A will be received by Service Consumer A, even when Service Consumer A is unavailable.

C- The Asynchronous Queuing pattern can be applied so that messaging queues are established between Service A and Service B and between Service Consumer A and Service A. This way, messages are never lost due to the unavailability of Service A or Service B.

D- The Event-Driven Messaging pattern can be applied by establishing a subscriber -publisher relationship between Service Consumer A and a database monitoring agent introduced through the application of the Service Agent pattern. The database monitoring agent monitors updates made by the legacy client to the database. This way, every time the data value is updated, an event is triggered and the database monitoring agent, acting as the publisher, can notify Service Consumer A, which acts as the subscriber.
The Asynchronous Queuing pattern can be applied between Service Consumer A and the database monitoring agent so that the event notification message sent out by the database monitoring agent will be received by Service Consumer A, even when Service Consumer A is unavailable.

Answer:

A

Explanation:

This solution is the most appropriate one among the options presented. By using the Event-Driven Messaging pattern, Service A can be notified of changes to the data value without having to be invoked repeatedly by Service Consumer A, which reduces the resources required for message exchange. Asynchronous Queuing ensures that the event notification message is not lost due to the unavailability of Service A or Service B. This approach improves the efficiency of the service composition architecture.