DUMPSBOSS.

SOA Design & Architecture Lab with Services & Microservices

SOA S90.08B

Version Demo

Total Demo Questions: 5

Total Premium Questions: 17

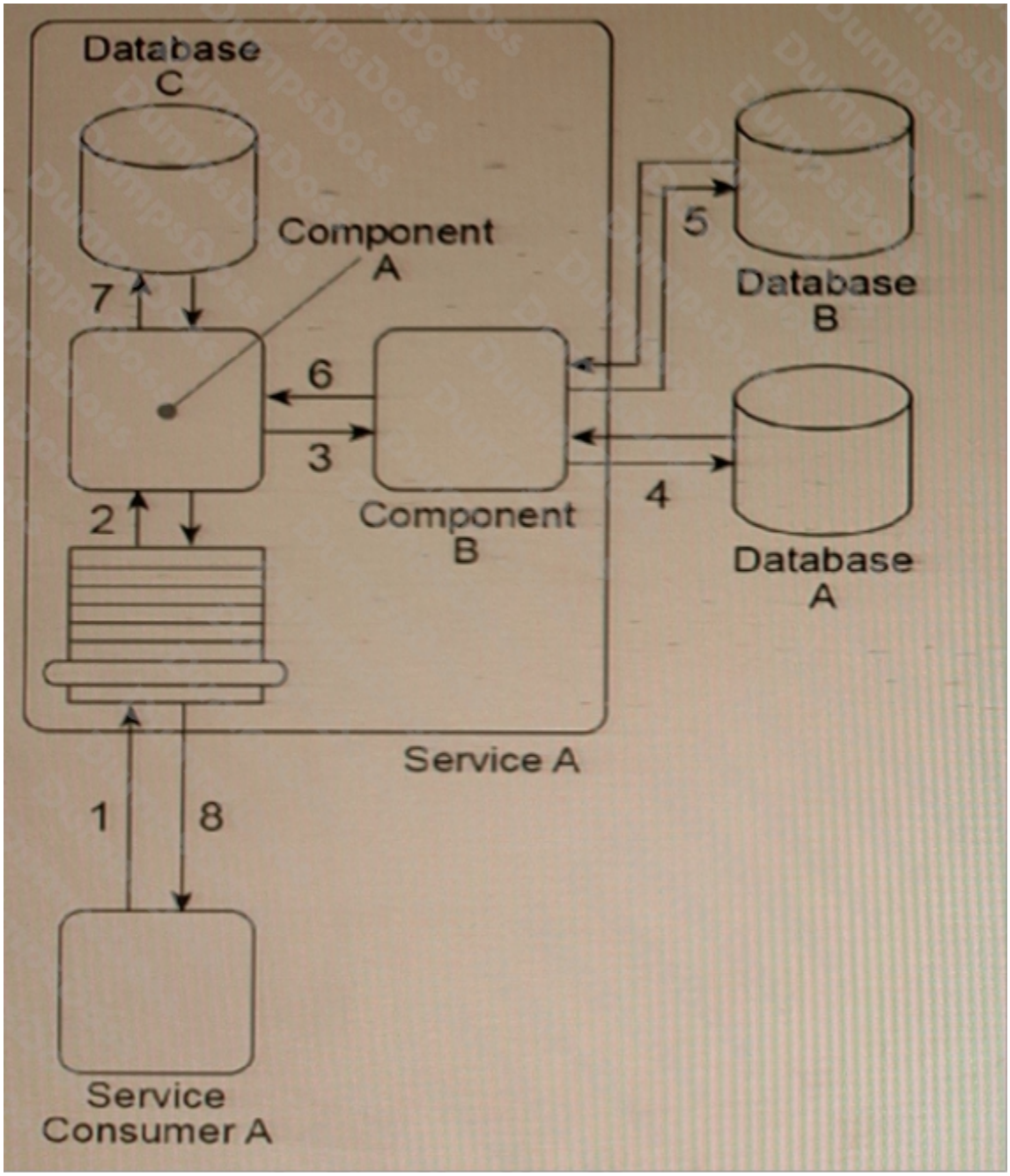
Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

QUESTION NO: 1



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?

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 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.

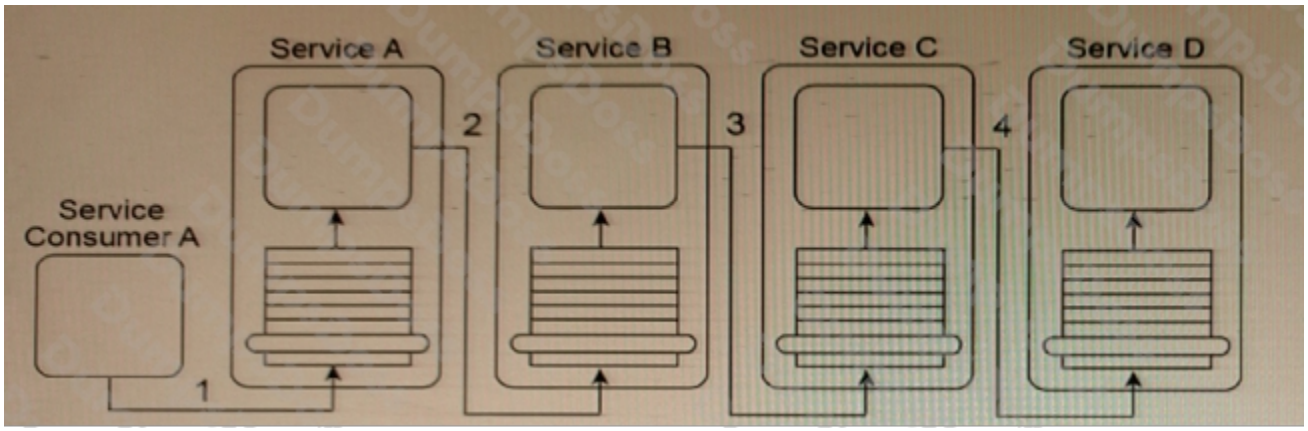
ANSWER: B

Explanation:

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

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 NO: 2



Service Consumer A sends a message to Service A (1), which then forwards the message to Service B (2). Service B forwards the message to Service C (3), which finally forwards the message to Service D (4). However, Services A, B and C each contain logic that reads the contents of the message to determine what intermediate processing to perform and which service to forward the message to. As a result, what is shown in the diagram is only one of several possible runtime scenarios.

Currently, this service composition architecture is performing adequately, despite the number of services that can be involved in the transmission of one message. However, you are told that new logic is being added to Service A that will require it to compose one other service to retrieve new data at runtime that Service A will need access to in order to determine where to forward the message to. The involvement of the additional service will make the service composition too large and slow.

What steps can be taken to improve the service composition architecture while still accommodating the new requirements and avoiding an increase in the amount of service composition members?

A. The Service Instance Routing pattern can be applied to introduce a Routing service to provide a centralized service to contain routing-related business rules. This new Routing service can be accessed by Service A and Service C so they can determine where to forward messages to at runtime. The Service Reusability principle can be further applied to ensure that the logic in all remaining services is designed to be multi-purpose and reusable.

B. The Asynchronous Queuing pattern can be applied together with a Routing service that is invoked by messages read from a messaging queue. This new Routing service can replace Service B and can be accessed by Service A and Service C so they can determine where to forward messages to at runtime. The Service Loose Coupling principle can be further applied to ensure that the new Routing service remains decoupled from other services so that it can perform its routing functions independently from service contract invocation.

C. The Intermediate Routing pattern can be applied together with the Service Agent pattern by removing Service B or Service C from the service composition and replacing it with a service agent capable of intercepting and forwarding the message at runtime based on pre-defined routing logic. The Service Discoverability principle can be further applied to ensure that Service A can be found by any future service consumers.

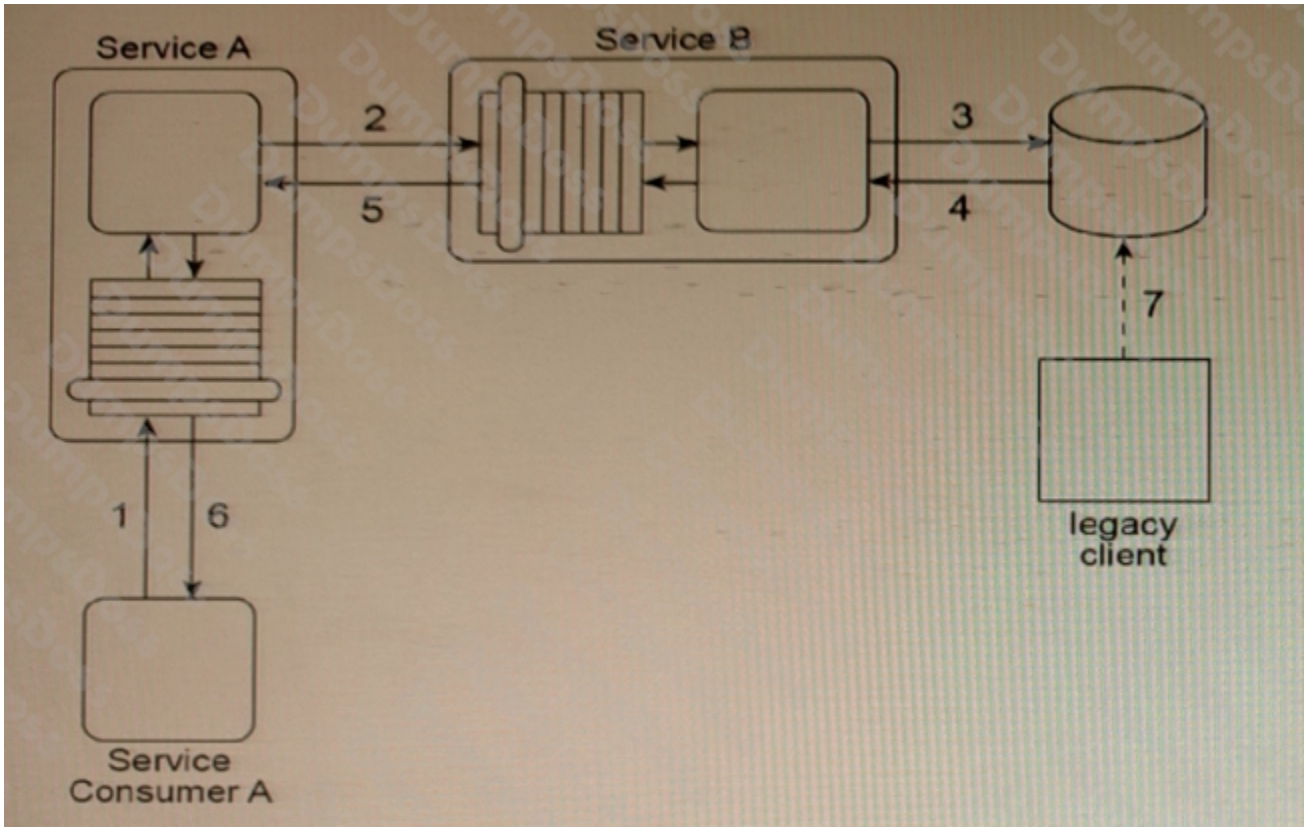
D. The Intermediate Routing pattern can be applied together with the Service Agent pattern to establish a service agent capable of intercepting and forwarding the message at runtime based on pre-defined routing logic. The Service Composability principle can be further applied to ensure that all services are designed as effective service composition participants.

ANSWER: D

Explanation:

This solution addresses the issue of the service composition becoming too large and slow by introducing a new Routing service that is invoked by messages read from a messaging queue. This allows Service A and Service C to determine where to forward messages to at runtime without the need for additional services in the composition. The Service Loose Coupling principle is applied to ensure that the new Routing service remains decoupled from other services so that it can perform its routing functions independently from service contract invocation.

QUESTION NO: 3



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?

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.

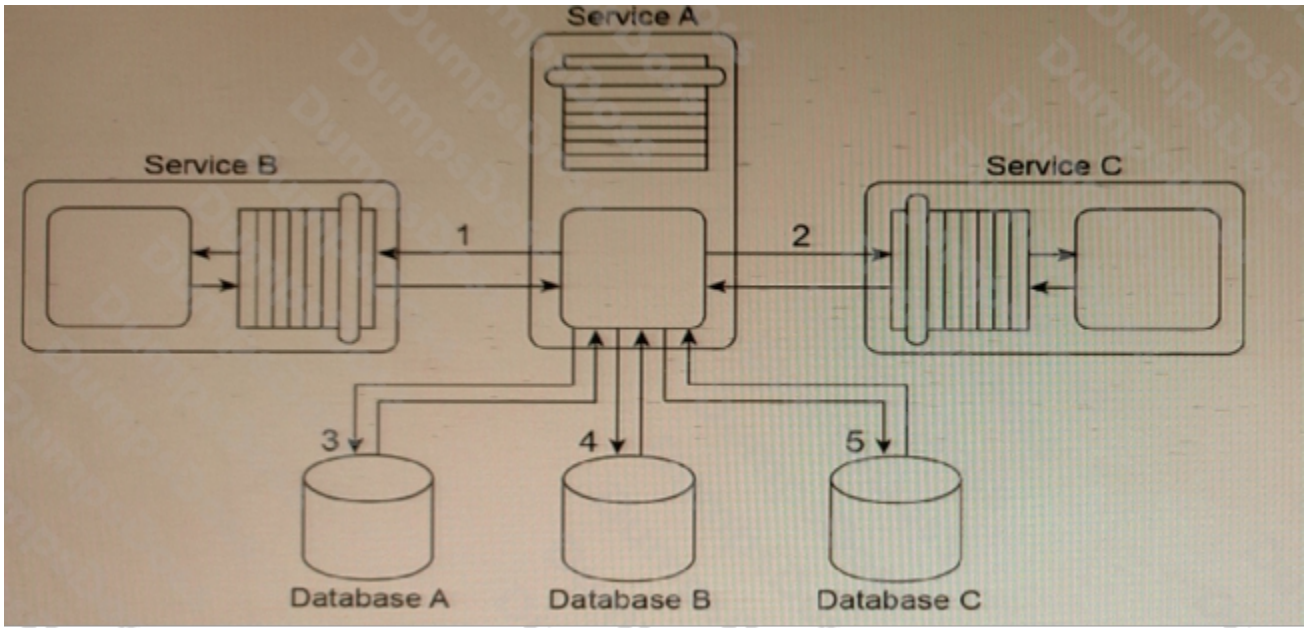
D. The Event-Driven Messaging pattern can be applied by establishing a subscriber -publisher relationship between Service Consumer A and a database monitoring agent introduced through the application of the Service Agent pattern. The database monitoring agent monitors updates made by the legacy client 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.

QUESTION NO: 4



Service A is an entity service that provides a set of generic and reusable service capabilities. In order to carry out the functionality of any one of its service capabilities, Service A is required to compose Service B (1) and Service C (2), and Service A is required to access Database A (3), Database B (4), and Database C (5). These three databases are shared by other applications within the IT enterprise.

All of service capabilities provided by Service A are synchronous, which means that for each request a service consumer makes, Service A is required to issue a response message after all of the processing has completed.

Service A is one of many entity services that reside In a highly normalized service Inventory. Because Service A provides agnostic logic, it is heavily reused and is currently part of many service compositions.

You are told that Service A has recently become unstable and unreliable. The problem has been traced to two issues with the current service architecture. First, Service B, which Is also an entity service, is being increasingly reused and has itself become unstable and unreliable. When Service B fails, the failure is carried over to Service A. Secondly, shared Database B has a complex data model. Some of the queries issued by Service A to shared Database B can take a very long time to complete.

What steps can be taken to solve these problems without compromising the normalization of the service inventory?

A. The Redundant Implementation pattern can be applied to Service A, thereby making duplicate deployments of the service available. This way, when one implementation of Service A is too busy, another implementation can be accessed by service consumers instead. The Service Data Replication pattern can be applied to establish a dedicated database that contains an exact copy of the data from shared Database B that is required by Service A.

B. The Redundant Implementation pattern can be applied to Service B, thereby making duplicate deployments of the service available. This way, when one implementation of Service B is too busy, another implementation can be accessed by Service A instead. The Data Model Transformation pattern can be applied to establish a dedicated database that contains an exact copy of the data from shared Database B that is required by Service A.

C. The Redundant Implementation pattern can be applied to Service B, thereby making duplicate deployments of the service available. This way, when one implementation of Service B is too busy, another implementation can be accessed by Service A instead. The Service Data Replication pattern can be applied to establish a dedicated database that contains a copy of the data from shared Database B that is required by Service A. The replicated database is designed with an optimized data model to improve query execution performance.

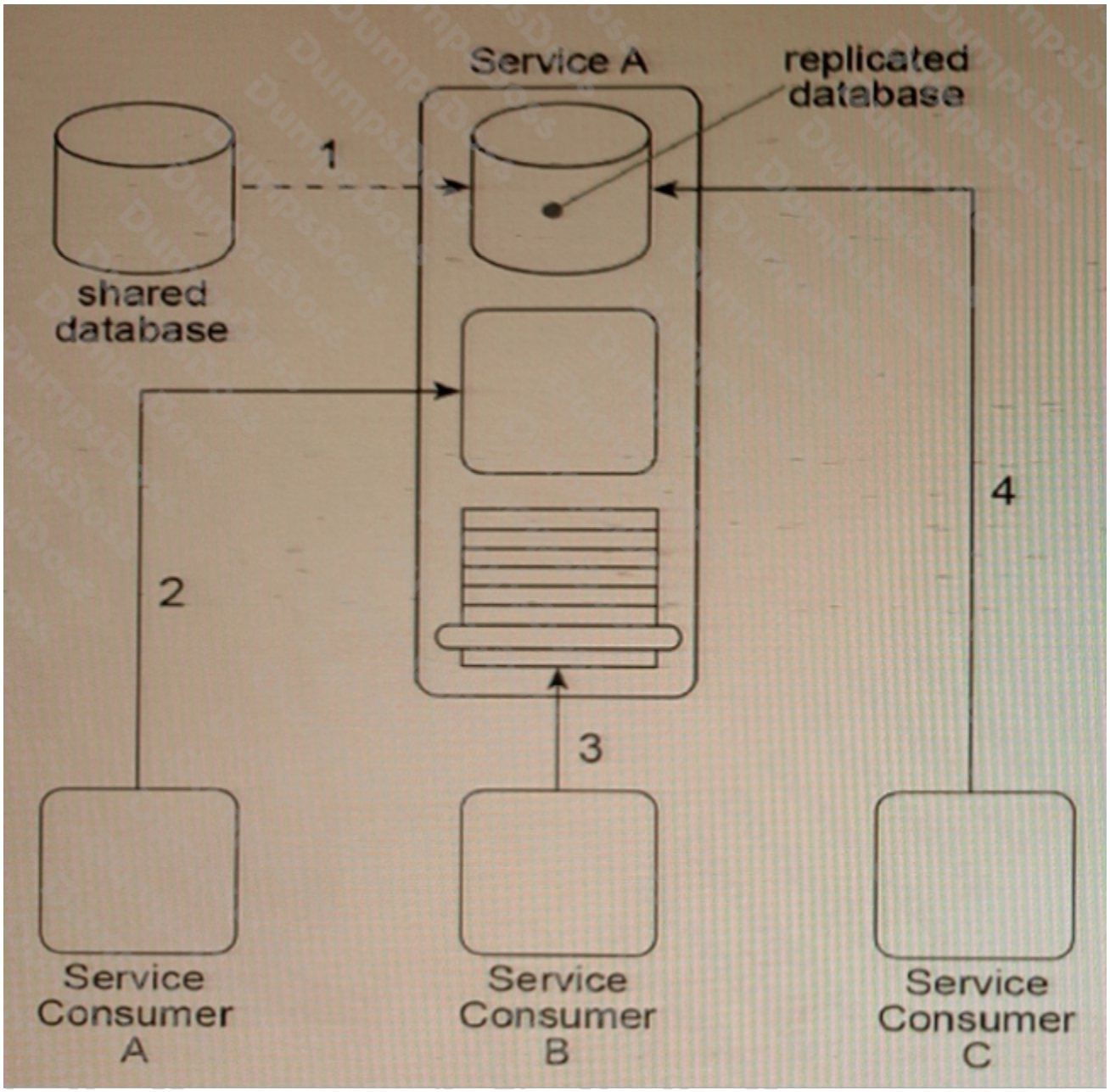
D. The Redundant Implementation pattern can be applied to Service A, thereby making duplicate deployments of the service available. This way, when one implementation of Service A is too busy, another implementation can be accessed by service consumers instead. The Service Statelessness principle can be applied with the help of the State Repository pattern In order to establish a state database that Service A can use to defer state data it may be required to hold for extended periods, thereby improving its availability and scalability.

ANSWER: C

Explanation:

This solution addresses both issues with the current service architecture. By applying the Redundant Implementation pattern to Service B, duplicate deployments of the service are made available, ensuring that when one implementation fails, another can be accessed by Service A. Additionally, the Service Data Replication pattern can be applied to establish a dedicated database that contains a copy of the data from shared Database B that is required by Service A. This replicated database is designed with an optimized data model to improve query execution performance, ensuring that queries issued by Service A to the database can complete more quickly, improving the overall stability and reliability of Service A. By applying these patterns, the problems with Service A can be solved without compromising the normalization of the service inventory.

QUESTION NO: 5



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?

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.