Arcitura Education

S90.08B Exam

SOA Design & Architecture Lab with Services & Microservices

Thank you for downloading S90.08B exam PDF Demo

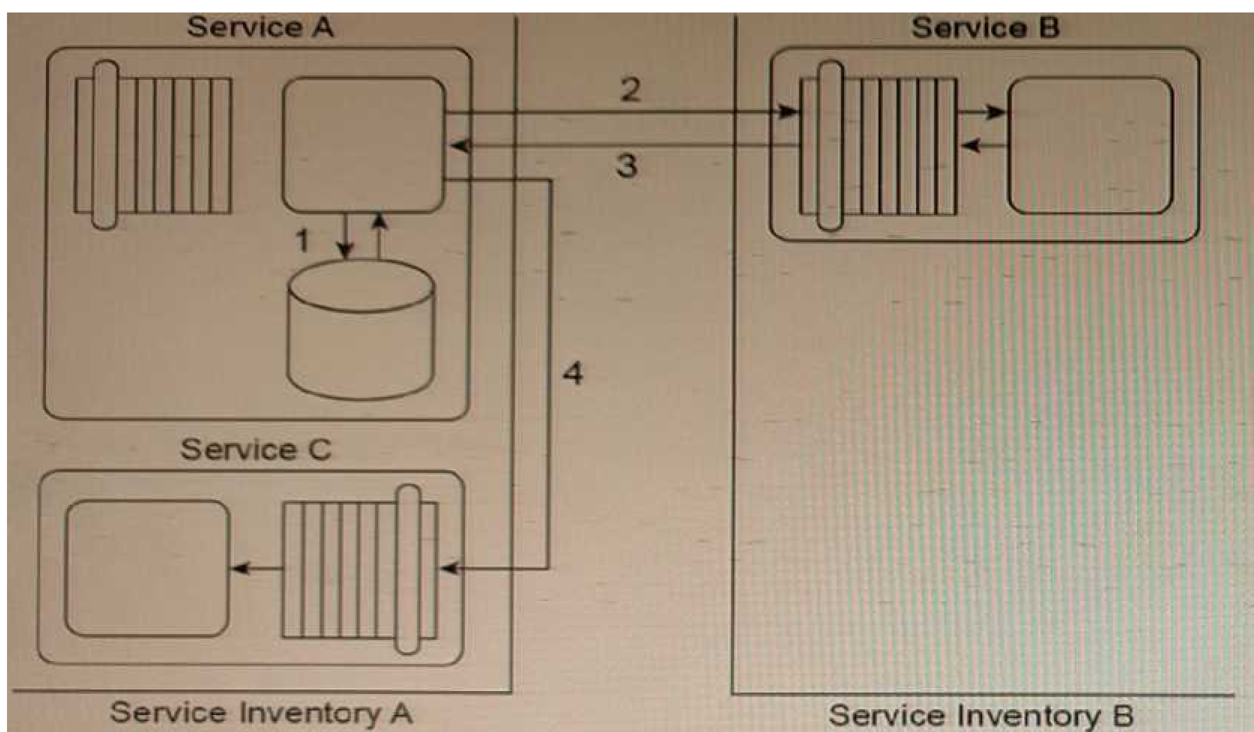
You can buy Latest S90.08B Full Version Download

https://www.certkillers.net/Exam/S90.08B

Version: 4.1

Question: 1

Refer to Exhibit.



Service A is a task service that sends Service B a message (2) requesting that Service B return data back to Service A in a response message (3). Depending on the response received, Service A may be required to send a message to Service C (4) for which it requires no response.

Before it contacts Service B, Service A must first retrieve a list of code values from its own database (1) and then place this data into its own memory. If it turns out that it must send a message to Service C, then Service A must combine the data it receives from Service B with the data from the code value list in order to create the message it sends to Service C. If Service A is not required to invoke Service C, it can complete its task by discarding the code values.

Service A and Service C reside in Service Inventory

A. Service B resides in Service Inventory B.

You are told that the services in Service Inventory A were designed with service contracts that are based on different design standards and technologies than the services In Service Inventory B. As a

result, Service A is a SOAP-based Web service and Service B Is a REST service that exchanges JSON-formatted messages. Therefore, Service A and Service B cannot currently communicate. Furthermore, Service C is an agnostic service that is heavily accessed by many concurrent service consumers. Service C frequently reaches its usage thresholds, during which it is not available and messages sent to it are not received.

What steps can be taken to solve these problems?

A. The Data Model Transformation pattern can be applied by establishing an intermediate processing layer between Service A and Service B that can transform a message from one data model to another at runtime. The Intermediate Routing and Service Agent patterns can be applied so that when Service B sends a response message, a service agent can intercept the message and, based on its contents, either forward the message to Service A or route the message to Service C. The Service Autonomy principle can be further applied to Service C together with the Redundant Implementation pattern to help establish a more reliable and scalable service architecture.

B. The Data Format Transformation pattern can be applied by establishing an intermediate processing layer between Service A and Service B that can transform a message from one data format to another at runtime. The Asynchronous Queuing pattern can be applied to establish an intermediate queue between Service A and Service C so that when Service A needs to send a message to Service C, the queue will store the message and retransmit it to Service C until it is successfully delivered. The Service Autonomy principle can be further applied to Service C together with the Redundant Implementation pattern to help establish a more reliable and scalable service architecture.

C. The Data Model Transformation pattern can be applied by establishing an intermediate processing layer between Service A and Service B that can transform a message from one data model to another at runtime. The Intermediate Routing and Service Agent patterns can be applied so that when Service B sends a response message, a service agent can intercept the message and, based on its contents, either forward the message to Service A or route the message to Service C. The Service Statelessness principle can be applied with the help of the State Repository pattern so that Service A can write the code value data to a state database while it is waiting for Service B to respond.

D. The Data Format Transformation pattern can be applied by establishing an intermediate processing layer between Service A and Service B that can transform a message from one data format to another at runtime. The Asynchronous Queuing pattern can be applied to establish an intermediate queue between Service A and Service B so that when Service A needs to send a message to Service B, the queue will store the message and retransmit it to Service B until it is successfully delivered. The Service Reusability principle can be further applied to Service C together with the Redundant Implementation pattern to help establish a more reusable and scalable service architecture.

Answer: B

Explanation:

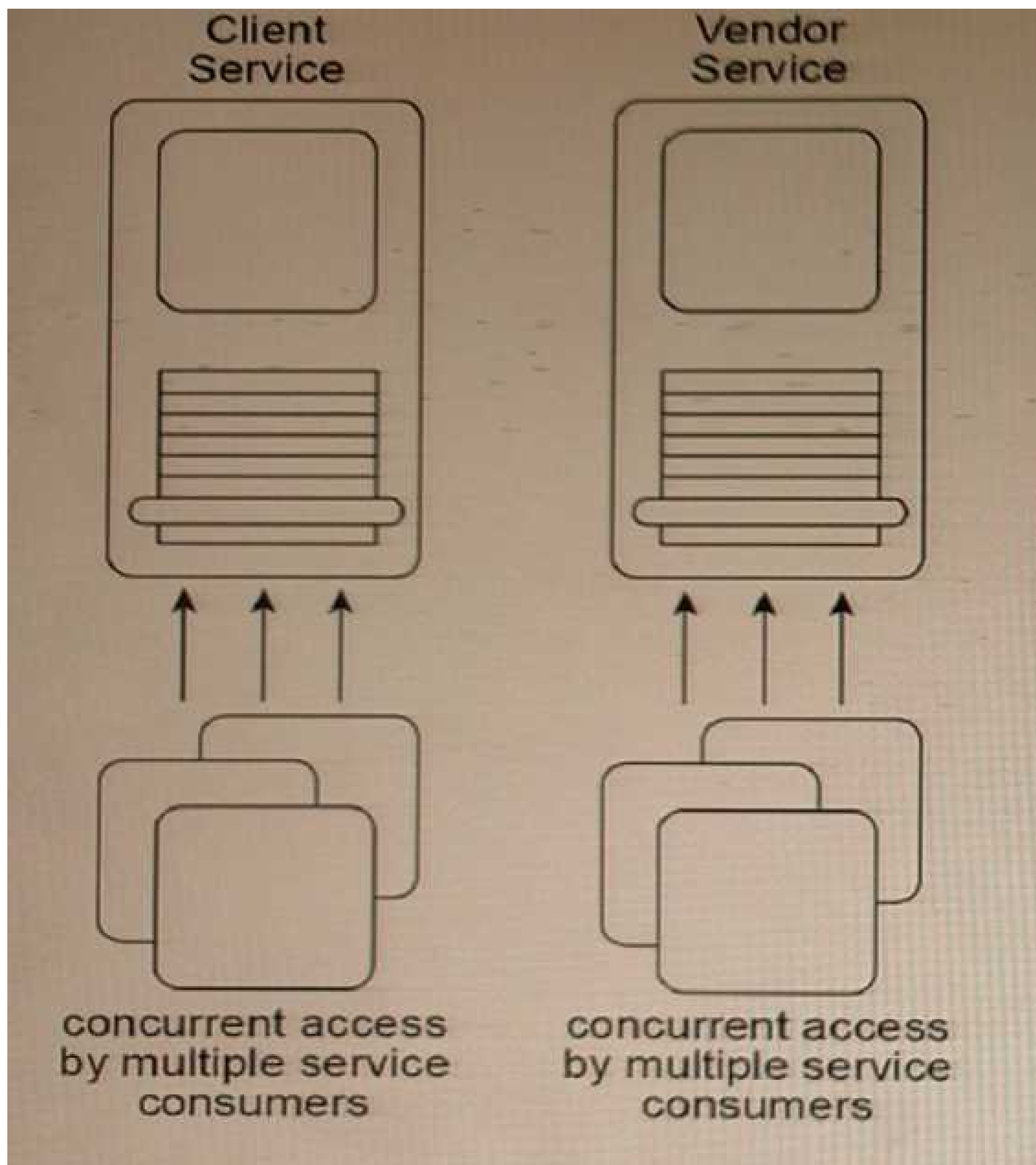
The problem is that Service A and Service B are using different technologies and cannot communicate. Therefore, an intermediate processing layer can be established that can transform messages from one data format to another at runtime. This can be achieved using the Data Format Transformation pattern.

Additionally, Service C frequently reaches its usage thresholds and is not always available, so an Asynchronous Queuing pattern can be applied to establish an intermediate queue between Service A and Service C. This queue will store the messages sent by Service A to Service C and retransmit them until they are successfully delivered. This approach improves the reliability of the system.

Moreover, the Redundant Implementation pattern can be applied to Service C to ensure its availability and scalability, and the Service Autonomy principle can be applied to make Service C independent of other services.

Question: 2

Refer to Exhibit.



The Client and Vendor services are agnostic services that are both currently part of multiple service compositions. As a result, these services are sometimes subjected to concurrent access by multiple service consumers.

The Client service primarily provides data access logic to a client database but also coordinates with other services to determine a clients credit rating. The Vendor service provides some data access logic but can also generate various dynamic reports based on specialized business requirements.

After reviewing historical statistics about the runtime activity of the two services, it is discovered that the Client service is serving an ever-increasing number of service consumers. It is regularly timing out, which in turn increases its call rate as service consumers retry their requests. The Vendor service

occasionally has difficulty meeting its service-level agreement (SLA) and when this occurs, penalties are assessed.

Recently, the custodian of the Client service was notified that the Client service will be made available to new service consumers external to its service inventory. The Client service will be providing free credit rating scores to any service consumer that connects to the service via the Internet. The Vendor service will remain internal to the service inventory and will not be exposed to external access.

Which of the following statements describes a solution that addresses these issues and requirements?

A. The API Gateway pattern, together with the Inventory Endpoint pattern, can be applied to the service inventory to establish an inventory endpoint service and an intermediary layer of processing that will be accessed by external service consumers and that will interact with the Client service to process external service consumer requests. The Redundant Implementation pattern can be applied to both the Client and Vendor services to increase their availability and scalability.

B. The Official Endpoint pattern can be applied to the Client service to establish a managed endpoint for consumption by service consumers external to the service inventory. The Concurrent Contracts pattern can be applied to the Vendor service, enabling it to connect with alternative Client service implementation, should the first attempt to connect fail.

C. The State Repository pattern can be applied to the Client and Vendor services to establish a central statement management database that can be used to overcome runtime performance problems. The Official Endpoint pattern can be further applied to increase the availability and scalability of the Client service for service consumers external to the service inventory.

D. The Microservice Deployment pattern is applied to the Client service to improve its autonomy and

responsiveness to a greater range of service consumers. The Containerization pattern is applied to the Vendor service to establish a managed environment with a high degree of isolation for its report-related processing. The Endpoint Redirection pattern is further applied to ensure that request messages from service consumers outside of the service inventory are redirected away from the Client service.

Answer: A

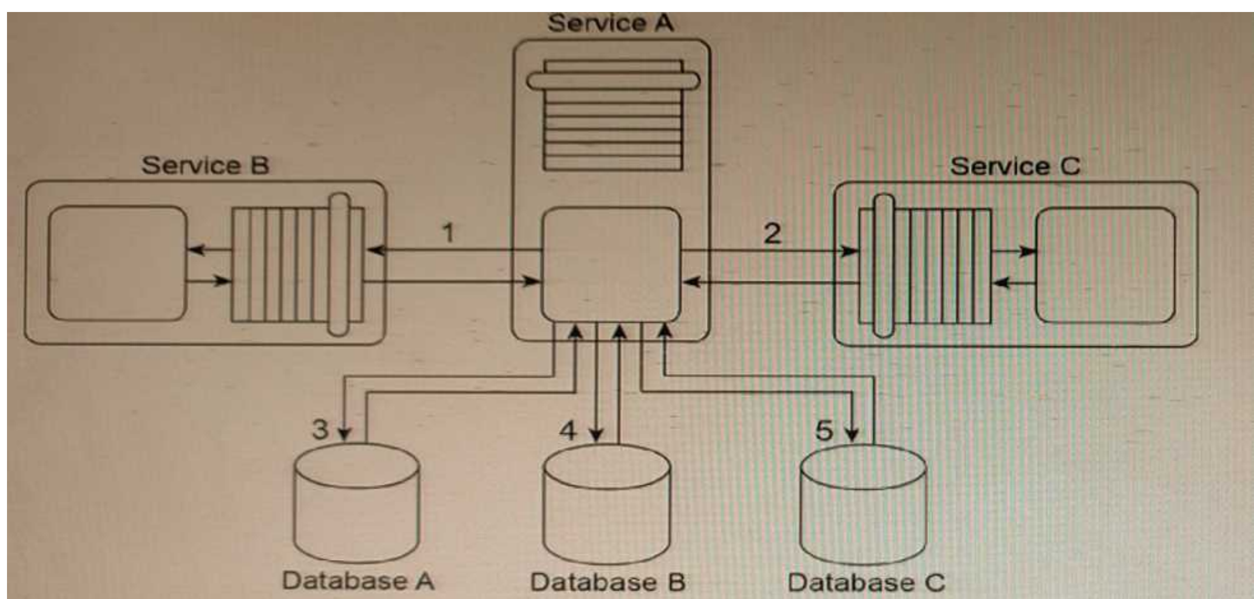
Explanation:

This solution addresses the specific requirements and issues identified in the scenario. The Official

Endpoint pattern can be applied to the Client service to establish a managed endpoint for consumption by service consumers external to the service inventory, which will allow for controlled and managed access to the service. The Concurrent Contracts pattern can be applied to the Vendor service, which will enable it to connect with alternative Client service implementation if the first attempt to connect fails, thereby increasing its availability and reducing the possibility of penalties being assessed due to not meeting its SLA.

Question: 3

Refer to Exhibit.



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

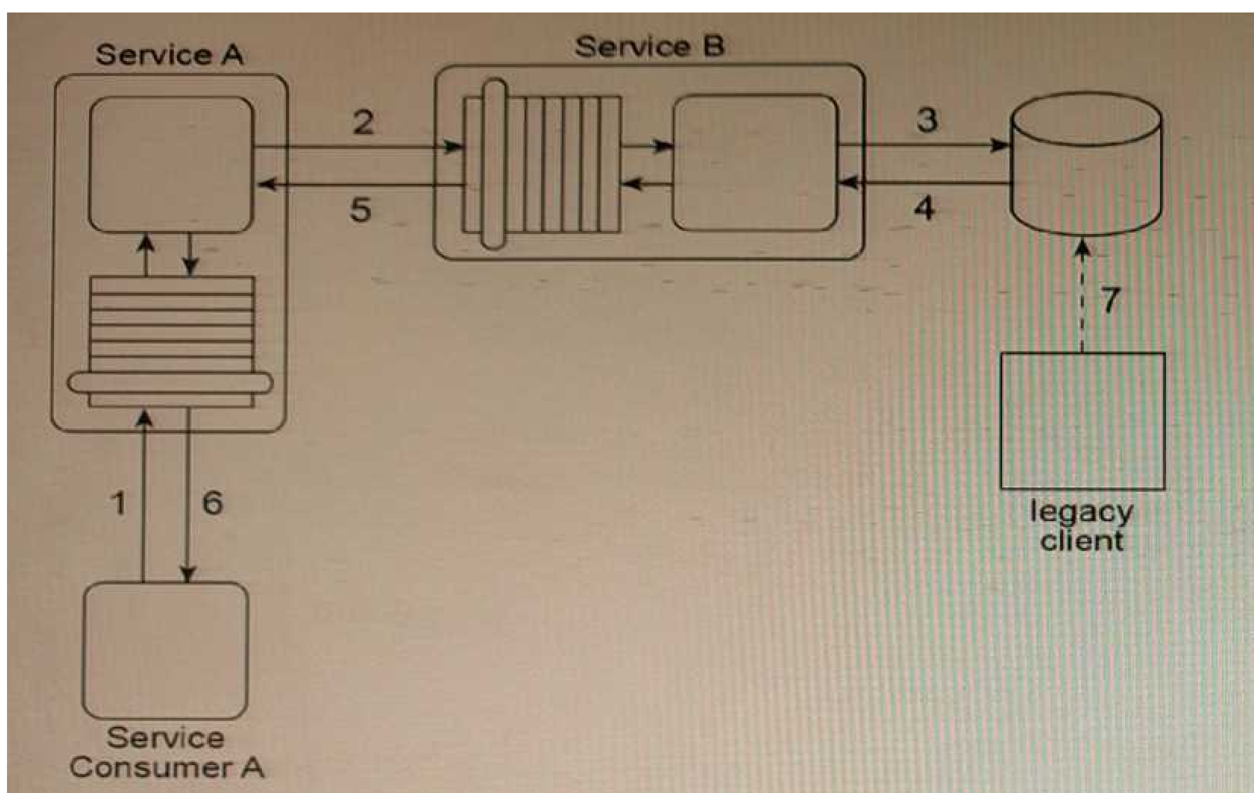
Explanation:

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: 4

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?

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.

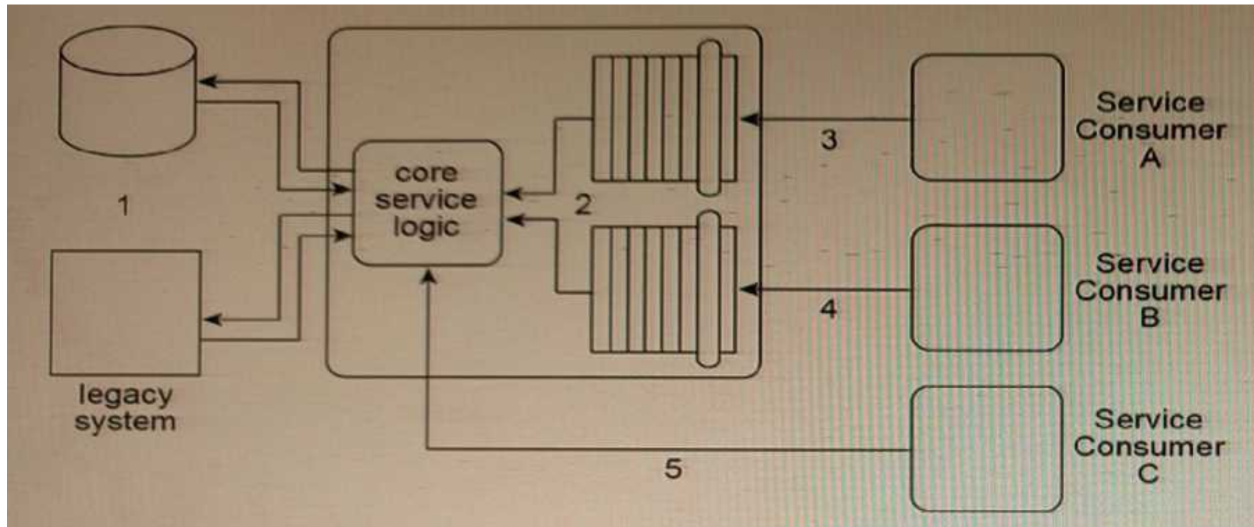
Answer: A

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: 5

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?

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.

Thank You for trying S90.08B PDF Demo

To Buy New S90.08B Full Version Download visit link below

https://www.certkillers.net/Exam/S90.08B

Start Your S90.08B Preparation

Use Coupon “CKNET” for Further discount on the purchase of Full Version Download. Test your S90.08B preparation with exam actual questions.