DUMPSARENA

Designing and Implementing Cloud-Native Applications Using Microsoft Azure Cosmos DB

Microsoft DP-420

Version Demo

Total Demo Questions: 10

Total Premium Questions: 92

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, New Update	41
Topic 2, Case Study 1	2
Topic 3, Case Study 2	2
Topic 4, Mixed Questions	47
Total	92

QUESTION NO: 1 - (HOTSPOT)

HOTSPOT

You have an Azure Cosmos DB Core (SQL) API account named storage1 that uses provisioned throughput capacity mode.

The storage1 account contains the databases shown in the following table.

Name	Throughput	Max request units per second (RU/s)	Geo-redundancy	Multi-region writes	Number of regions
db1	Autoscale	5,000	Disabled	Disabled	1
db2	Autoscale	8,000	Enabled	Enabled	3

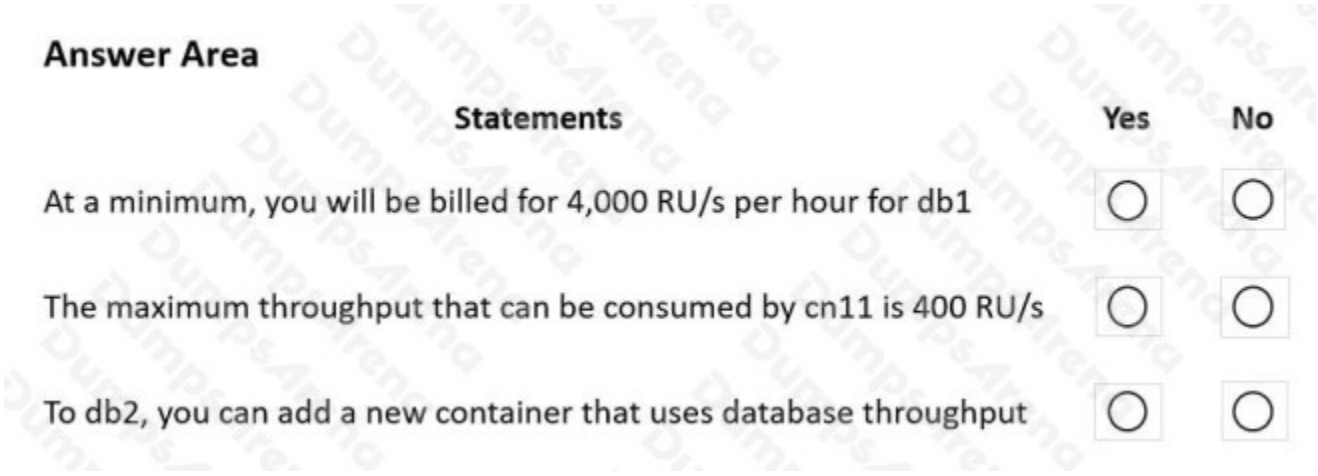
The databases contain the containers shown in the following table.

Name	Database	Throughput
cn01	db1	Container - autoscale maximum RU/s of 10,000
cn02	db1	Database
cn03	db1	Database
cn04	db1	Database
cn05	db1	Database
cn11	db2	Database
cn12	db2	Database
cn13	db2	Database
cn14	db2	Database
cn15	db2	Database
cn16	db2	Database
cn17	db2	Database
cn18	db2	Database

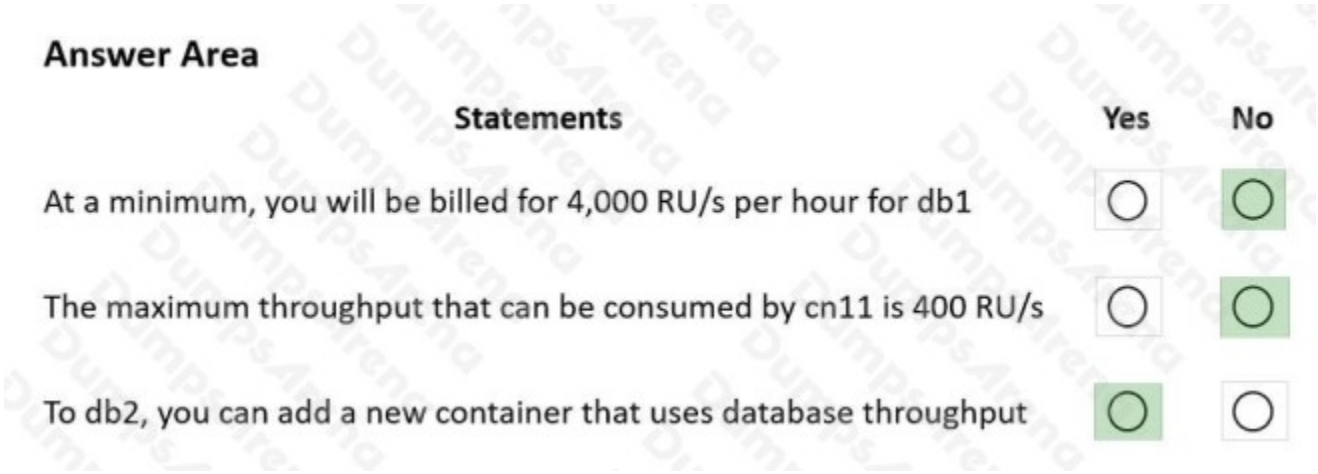
For each of the following statements, select Yes if the statement is true. Otherwise, select No.

NOTE: Each correct selection is worth one point.

Hot Area:



ANSWER:



Explanation:

Box 1: No

Four containers with 1000 RU/s each.

Box 2: No

Max 8000 RU/s for db2. 8 containers, so 1000 RU/s for each container.

Box 3: Yes

Max 8000 RU/s for db2. 8 containers, so 1000 RU/s for each container. Can very well add an additional container.

Reference: https://docs.microsoft.com/en-us/azure/cosmos-db/plan-manage-costs https://azure.microsoft.com/en-us/pricing/details/cosmos-db/

QUESTION NO: 2

You have a container named container1 in an Azure Cosmos DB for NoSQL account named account1 that is set to the session default consistency level. The average size of an item in container1 is 20 KB.

You have an application named App1 that uses the Azure Cosmos DB SDK and performs a point read on the same set of items in container1 every minute.

You need to minimize the consumption of the request units (RUs) associated to the reads by App1. What should you do?

A. In account1, change the default consistency level to bounded staleness.

B. In App1, change the consistency level of read requests to consistent prefix.

C. In account1, provision a dedicated gateway and integrated cache

D. In App1, modify the connection policy settings.

ANSWER: B

Explanation:

The cost of a point read for a 1 KB item is 1 RU. The cost of other operations depends on factors such as item size, indexing policy, consistency level,and query complexity1. To minimize the consumption of RUs, you can optimize these factors according to your application needs.

For your scenario, one possible way to minimize the consumption of RUs associated to the reads by App1 is to change the consistency level of read requests to consistent prefix. Consistent prefix is a lower consistency level than session, which is the default consistency level for Azure Cosmos DB. Lower consistency levels consume fewer RUs than higher consistency levels2. Consistent prefix guarantees that reads never see out-of-order writes and that monotonic reads are preserved1. This may be suitable for your application if you can tolerate some eventual consistency.

QUESTION NO: 3 - (HOTSPOT)

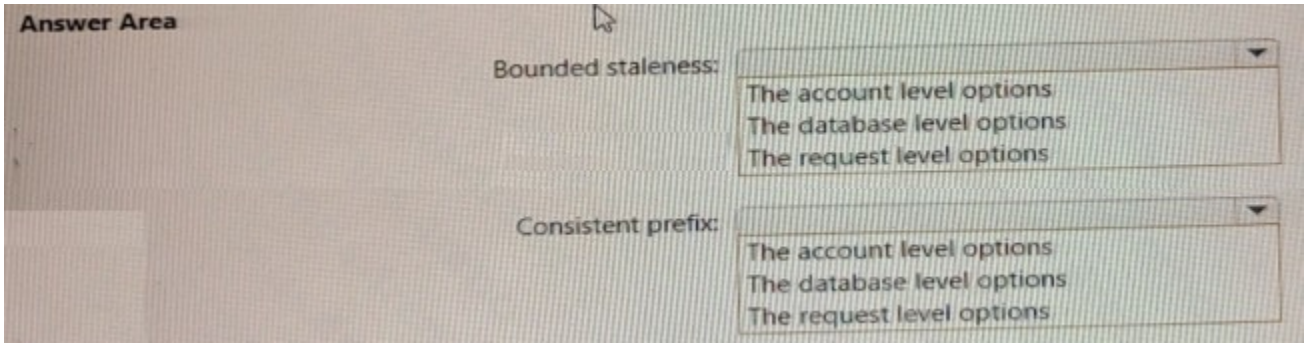
You have an Azure Cosmos DB account named account1 that has a default consistency level of session.

You have an app named App1.

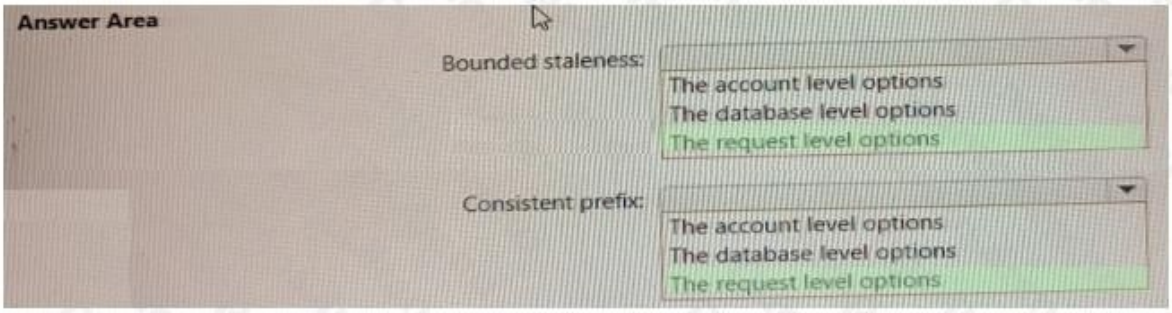
You need to ensure that the read operations of App1 can request either bounded staleness or consistent prefix consistency.

What should you modify for each consistency level? To answer, select the appropriate options in the answer area.

NOTE: Each correct selection is worth one point.



ANSWER:



Explanation:

Box 1 = The request level options

Azure Cosmos DB offers five well-defined consistency levels: strong, bounded staleness, session, consistent prefix and eventual. You can configure the default consistency level on your Azure Cosmos DB account at any time2. The default consistency level applies to all databases and containers under that account1. You can also override the default consistency level for a specific request by using the request options2.

Box 2 = The request level options

To modify the consistency level of a read operation in Azure Cosmos DB, you can use request-level options to override the account's default consistency setting. Therefore, to ensure that the read operations of App1 can request either consistent prefix or session consistency, you need to modify the request-level options for each operation. Reference: - https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels

QUESTION NO: 4

The following is a sample of a document in orders.

```
{
    "orderId" : "d4a91979b-5ead-43a3-b851-add9a71ac4b6",
    "customerId" : "f6e39103-bdc7-4346-9cfb-45daa4b2becf",
    "orderDate" : "2021-09-29",
 "orderItems" : [
   {
    "itemId" : "6c30412f-3cd7-4cab-813c-05942345720d",
    "name" : "blue pen",
    "type" : "pens",
    "count" : 10,
   },
   ...
 ],
 "total" : 12345,
 "status" : "ordered"
}
```

The orders container uses customerId as the partition key.

You need to provide a report of the total items ordered per month by item type. The solution must meet the following requirements:

- Ensure that the report can run as quickly as possible.
- Minimize the consumption of request units (RUs).

What should you do?

A. Configure the report to query orders by using a SQL query.

B. Configure the report to query a new aggregate container. Populate the aggregates by using the change feed.

C. Configure the report to query orders by using a SQL query through a dedicated gateway.

D. Configure the report to query a new aggregate container. Populate the aggregates by using SQL queries that run daily.

ANSWER: B

Explanation:

You can facilitate aggregate data by using Change Feed and Azure Functions, and then use it for reporting.

Reference: https://docs.microsoft.com/en-us/azure/cosmos-db/change-feed

QUESTION NO: 5

You have an Azure Cosmos DB Core (SQL) API account that is used by 10 web apps.

You need to analyze the data stored in the account by using Apache Spark to create machine learning models. The solution must NOT affect the performance of the web apps.

Which two actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. In an Apache Spark pool in Azure Synapse, create a table that uses cosmos.olap as the data source.

B. Create a private endpoint connection to the account.

C. In an Azure Synapse Analytics serverless SQL pool, create a view that uses OPENROWSET and the CosmosDB provider.

D. Enable Azure Synapse Link for the account and Analytical store on the container.

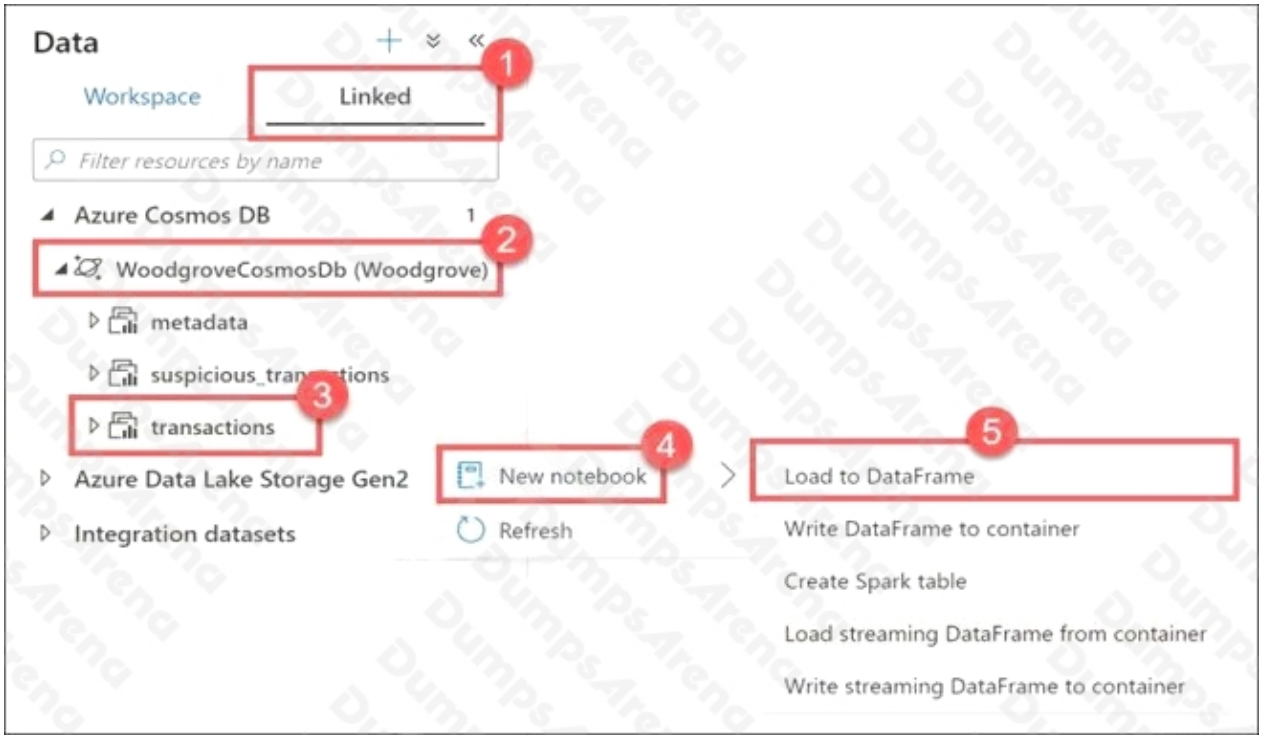
E. In an Apache Spark pool in Azure Synapse, create a table that uses cosmos.oltp as the data source.

ANSWER: A D

Explanation:

Explore analytical store with Apache Spark 1. Navigate to the Data hub.

2. Select the Linked tab (1), expand the Azure Cosmos DB group (if you don't see this, select the Refresh button above), then expand the WoodgroveCosmosDb account (2). Right-click on the transactions container (3), select New notebook (4), then select Load to DataFrame (5).



3. In the generated code within Cell 1 (3), notice that the spark.read format is set to cosmos.olap. This instructs Synapse Link to use the container's analytical store. If we wanted to connect to the transactional store, like to read from the change feed or write to the container, we'd use cosmos.oltp instead.

Reference: https://github.com/microsoft/MCW-Cosmos-DB-Real-Time-Advanced-Analytics/blob/main/Hands-on%20lab/HOL%20step-by%20step%20-%20Cosmos%20DB%20real-time%20advanced%20analytics.md

QUESTION NO: 6

You need to configure an Apache Kafka instance to ingest data from an Azure Cosmos DB Core (SQL) API account. The data from a container named telemetry must be added to a Kafka topic named iot. The solution must store the data in a compact binary format.

Which three configuration items should you include in the solution? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. "connector.class": "com.azure.cosmos.kafka.connect.source.CosmosDBSourceConnector"

B. "key.converter": "org.apache.kafka.connect.json.JsonConverter"

C. "key.converter": "io.confluent.connect.avro.AvroConverter"

D. "connect.cosmos.containers.topicmap": "iot#telemetry"

E. "connect.cosmos.containers.topicmap": "iot"

F. "connector.class": "com.azure.cosmos.kafka.connect.source.CosmosDBSinkConnector"

ANSWER: C D F

Explanation:

C: Avro is binary format, while JSON is text.

F: Kafka Connect for Azure Cosmos DB is a connector to read from and write data to Azure Cosmos DB. The Azure Cosmos DB sink connector allows you to export data from Apache Kafka topics to an Azure Cosmos DB database. The connector polls data from Kafka to write to containers in the database based on the topics subscription.

D: Create the Azure Cosmos DB sink connector in Kafka Connect. The following JSON body defines config for the sink connector. Extract:

"connector.class": "com.azure.cosmos.kafka.connect.sink.CosmosDBSinkConnector",

"key.converter": "org.apache.kafka.connect.json.AvroConverter" "connect.cosmos.containers.topicmap": "hotels#kafka"

Incorrect Answers:

B: JSON is plain text.

Note, full example:

{

"name": "cosmosdb-sink-connector",

"config": {

"connector.class": "com.azure.cosmos.kafka.connect.sink.CosmosDBSinkConnector",

"tasks.max": "1",

"topics": [

"hotels"

],

"value.converter": "org.apache.kafka.connect.json.AvroConverter",

"value.converter.schemas.enable": "false",

"key.converter": "org.apache.kafka.connect.json.AvroConverter",

"key.converter.schemas.enable": "false",

"connect.cosmos.connection.endpoint": "https://.documents.azure.com:443/",

"connect.cosmos.master.key": "",

"connect.cosmos.databasename": "kafkaconnect",

"connect.cosmos.containers.topicmap": "hotels#kafka"

} }

Reference: https://docs.microsoft.com/en-us/azure/cosmos-db/sql/kafka-connector-sink https://www.confluent.io/blog/kafka-connect-deep-dive-converters-serialization-explained/

QUESTION NO: 7

You have an Azure Cosmos DB for NoSQL account named account1 that supports an application named App1. App1 uses the consistent prefix consistency level.

You configure account1 to use a dedicated gateway and integrated cache.

You need to ensure that App1 can use the integrated cache.

Which two actions should you perform for APP1? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Change the connection mode to direct

B. Change the account endpoint to https://account1.sqlx.cosmos.azure.com.

C. Change the consistency level of requests to strong.

D. Change the consistency level of requests to session.

E. Change the account endpoint to https://account1.documents.azure.com

ANSWER: B D

Explanation:

the Azure Cosmos DB integrated cache is an in-memory cache that is built-in to the Azure Cosmos DB dedicated gateway. The dedicated gateway is a front-end compute that stores cached data and routes requests to the backend database. You can choose from a variety of dedicated gateway sizes based on the number of cores and memory needed for your workload1. The integrated cache can reduce the RU consumption and latency of read operations by serving them from the cache instead of the backend containers2.

For your scenario, to ensure that App1 can use the integrated cache, you should perform these two actions:

QUESTION NO: 8

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.

After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.

You have a container named container1 in an Azure Cosmos DB Core (SQL) API account.

You need to make the contents of container1 available as reference data for an Azure Stream Analytics job.

Solution: You create an Azure Data Factory pipeline that uses Azure Cosmos DB Core (SQL) API as the input and Azure Blob Storage as the output.

Does this meet the goal?

A. Yes

B. No

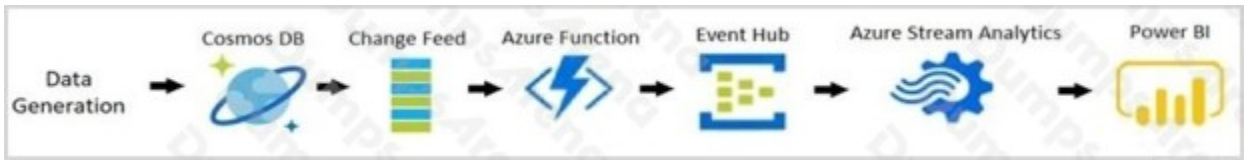
ANSWER: B

Explanation:

Instead create an Azure function that uses Azure Cosmos DB Core (SQL) API change feed as a trigger and Azure event hub as the output.

The Azure Cosmos DB change feed is a mechanism to get a continuous and incremental feed of records from an Azure Cosmos container as those records are being created or modified. Change feed support works by listening to container for any changes. It then outputs the sorted list of documents that were changed in the order in which they were modified.

The following diagram represents the data flow and components involved in the solution:



Reference: https://docs.microsoft.com/en-us/azure/cosmos-db/sql/changefeed-ecommerce-solution

QUESTION NO: 9

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.

After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.

You have a container named container1 in an Azure Cosmos DB Core (SQL) API account.

You need to make the contents of container1 available as reference data for an Azure Stream Analytics job.

Solution: You create an Azure function that uses Azure Cosmos DB Core (SQL) API change feed as a trigger and Azure event hub as the output.

Does this meet the goal?

A. Yes

B. No

ANSWER: A

Explanation:

The Azure Cosmos DB change feed is a mechanism to get a continuous and incremental feed of records from an Azure Cosmos container as those records are being created or modified. Change feed support works by listening to container for any changes. It then outputs the sorted list of documents that were changed in the order in which they were modified.

The following diagram represents the data flow and components involved in the solution:

Reference: https://docs.microsoft.com/en-us/azure/cosmos-db/sql/changefeed-ecommerce-solution

QUESTION NO: 10 - (DRAG DROP)

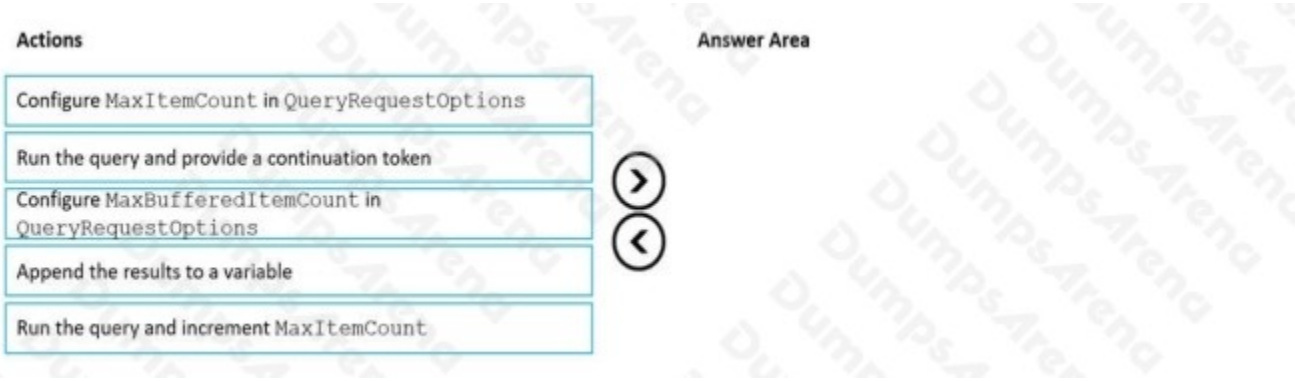
DRAG DROP

You have an app that stores data in an Azure Cosmos DB Core (SQL) API account The app performs queries that return large result sets.

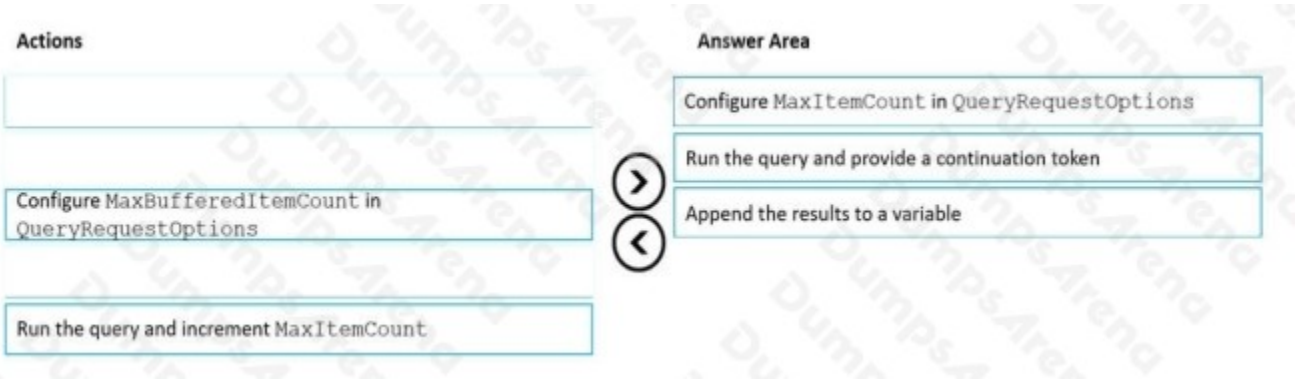
You need to return a complete result set to the app by using pagination. Each page of results must return 80 items.

Which three actions should you perform in sequence? To answer, move the appropriate actions from the list of actions to the answer area and arrange them in the correct order.

Select and Place:



ANSWER:



Explanation:

Step 1: Configure the MaxItemCount in QueryRequestOptions

You can specify the maximum number of items returned by a query by setting the MaxItemCount. The MaxItemCount is specified per request and tells the query engine to return that number of items or fewer.

Box 2: Run the query and provide a continuation token

In the .NET SDK and Java SDK you can optionally use continuation tokens as a bookmark for your query's progress. Azure Cosmos DB query executions are stateless at the server side and can be resumed at any time using the continuation token.

If the query returns a continuation token, then there are additional query results. Step 3: Append the results to a variable

Reference:

https://docs.microsoft.com/en-us/azure/cosmos-db/sql/sql-query-pagination