Vendor: Microsoft

Exam Code: AI-200

Exam Name: Azure AI Cloud Developer Associate

Version: DEMO

QUESTION 1
You configure ACR Tasks to automate image builds.

Container images must rebuild when:

```
- Application updates occur.
- Base image updates occur, such as when the underlying OS image is
updated.
- Regular scheduled rebuilds are required.
```

You need to configure ACR Tasks to support automated image rebuilds.

Which three triggers should you configure? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Timer trigger
B. Source code commit trigger
C. Registry event trigger
D. Base image update trigger
E. Webhook notification trigger

Answer: ABD
Explanation:
To support automated container image rebuilds based on your requirements, you need to configure source triggers, base image triggers, and timer triggers in Azure Container Registry (ACR) Tasks.

Timer Triggers: Automates regular scheduled rebuilds. This uses a cron schedule expression to run image builds at specific intervals (e.g., weekly or monthly) regardless of code changes.

Source Triggers: Automates rebuilds when application updates occur. This monitors changes in your source code repository (like GitHub or Azure Repos) and fires a build when code is committed.

Base Image Triggers: Automates rebuilds when the underlying OS or framework image updates. ACR tracks dependencies and automatically kicks off a new build when your defined FROM image changes in the public registry or your private registry.

Reference:
https://oneuptime.com/blog/post/2026-02-16-how-to-set-up-acr-tasks-for-automated-container-image-builds-on-git-commit/view

QUESTION 2
Drag and Drop Question

You are developing a .NET application that uses Azure Cosmos DB for NoSQL to store application data.

The application uses the Azure Cosmos DB for NoSQL SDK to interact with the database account.

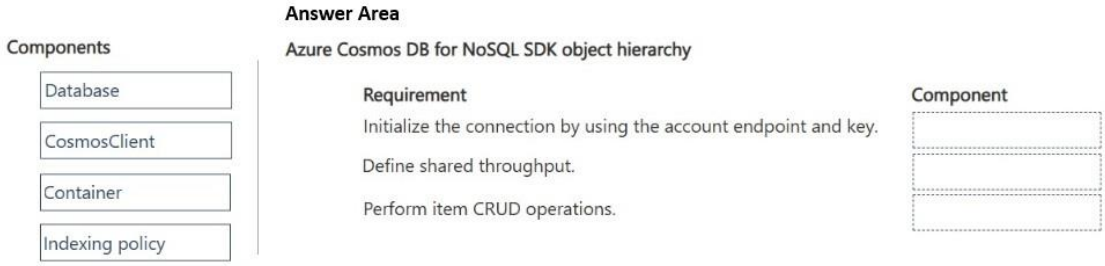
The application must perform the following tasks:

```
- Initialize the connection by using the account endpoint and key.
- Define shared throughput.
- Perform create, read, update, and delete (CRUD) operations on items
stored in a container.
```

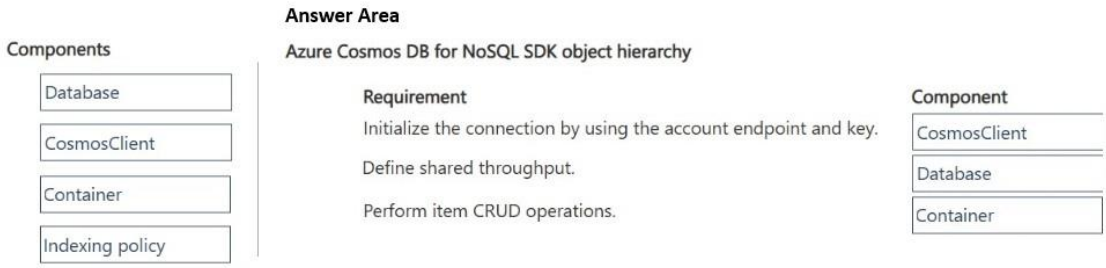
You need to implement the SDK components required for the application to access and manage data in Azure Cosmos DB for NoSQL.

Which SDK components should you use? To answer, move the appropriate components to the correct requirements. You may use each component once, more than once, or not at all. You may need to move the split bar between panes or scroll to view content.

NOTE: Each correct selection is worth one point.



Answer:



Explanation:
Box 1: CosmosClient
To initialize the connection to an Azure Cosmos DB for NoSQL account using the account endpoint and key, you must use the CosmosClient class.

Box 2: Database
The database SDK component should be used to define shared throughput.

In Azure Cosmos DB, shared throughput (provisioned Request Units per second or RU/s) is configured at the database level. When throughput is provisioned on a database, that capacity is shared among all the containers created within that specific database.

Box 3: Container
To perform item create, read, update, and delete (CRUD) operations, you should use the Container SDK component

In the Azure Cosmos DB for NoSQL SDK for .NET, individual JSON documents (items) live inside a container. The Container class exposes the specific methods required to execute CRUD operations on these items.

Reference:
https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-dotnet-get-started

QUESTION 3
Drag and Drop Question

You are developing a new page for a website that uses Azure Cosmos DB for data storage. The feature uses documents that have the following format:

```
{
  "name": "John",
  "city": "Seattle"
}
```

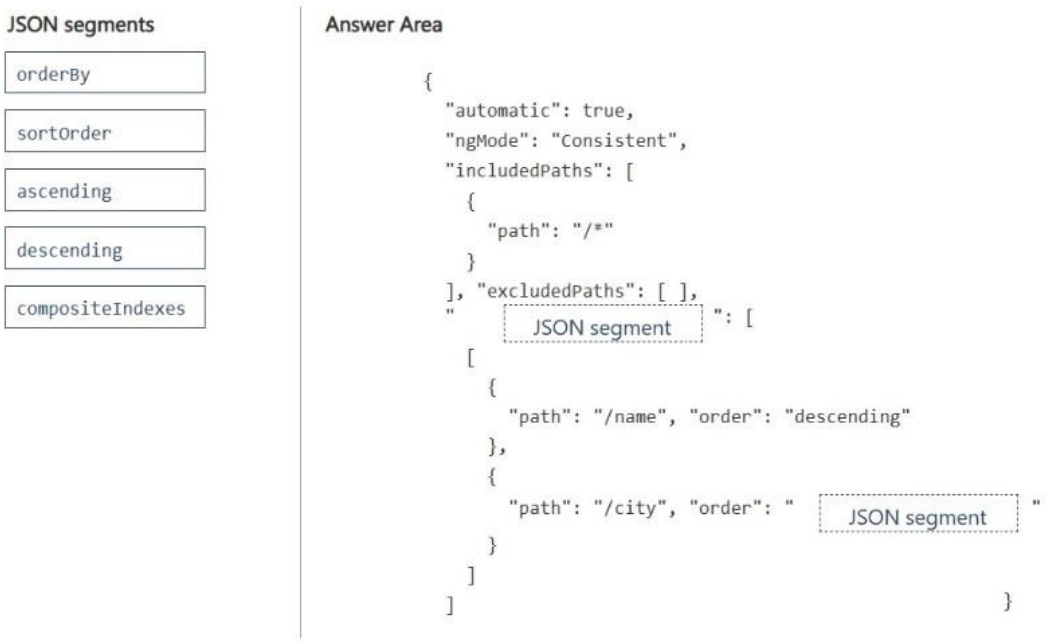
You must display data for the new page in a specific order. You create the following query for the page:

```
SELECT *
FROM People p
ORDER BY p.name, p.city DESC
```

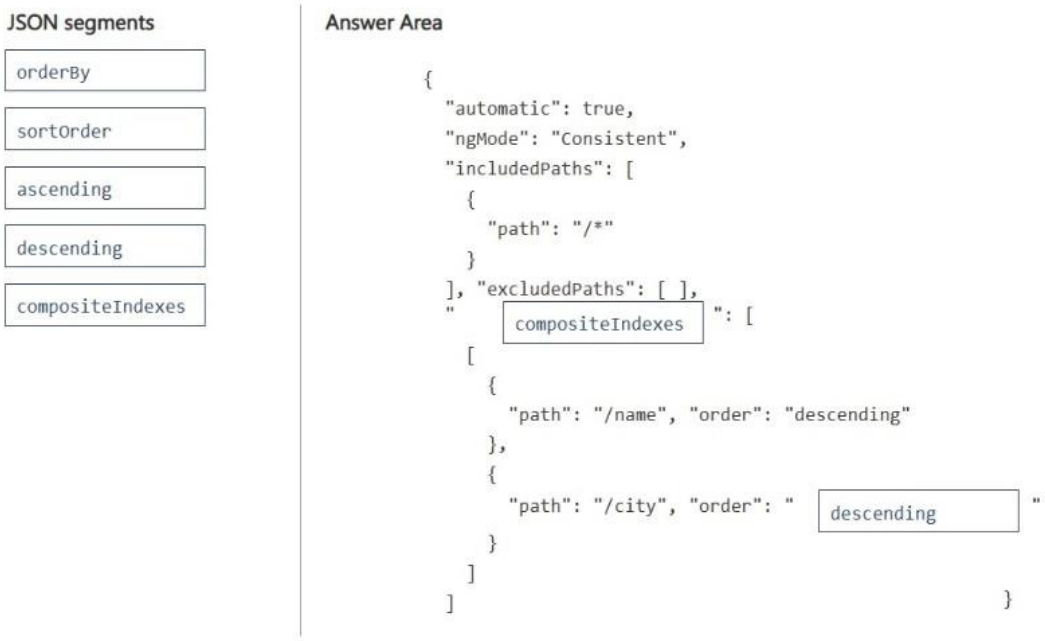
You need to configure an Azure Cosmos DB policy to support the query.

How should you configure the policy? To answer, drag the appropriate JSON segments to the correct locations. Each JSON segment may be used once, more than once, or not at all. You may need to drag the split bar between panes or scroll to view content.

NOTE: Each correct selection is worth one point.



Answer:



Explanation:
Box 1: compositeIndexes
Azure Cosmos DB requires a defined composite index for any query utilizing an ORDER BY clause with multiple properties.

Box 2: descending
In an ORDER BY clause, properties without an explicit direction default to ascending order. Your query ORDER BY p.name, p.city DESC translates to an ascending sort on /name and a descending sort on /city.

Reference:
https://docs.azure.cn/en-us/cosmos-db/index-policy

QUESTION 4
An ACA app processes messages from an Azure Storage queue.

The app must scale automatically based on messages in a specific Azure Storage queue by using a Kubernetes Event-driven Autoscaler (KEDA) custom scale rule.

You need to configure the required scale rule values.

Which two values should you configure? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Queue name
B. Polling interval
C. Trigger type

D. Maximum replicas

Answer: AC
Explanation:
[A]
QeueName: The exact name of the specific Azure Storage queue your app is polling is required.

[C]
Why Trigger Type is Required
When configuring a custom scale rule in Azure Container Apps (ACA), the platform utilizes KEDA underneath. Because a custom rule can connect to many different event sources (such as Kafka, Redis, or Azure Storage), you must explicitly define the Trigger type (e.g., azure-queue) so KEDA knows which specific scaler to initiate.

Reference:
https://techcommunity.microsoft.com/blog/fasttrackforazureblog/container-apps-a-practical-scaling-with-azure-queue-scale-rule/3722075

QUESTION 5
You have an Event Grid subscription that triggers an Azure Function.

You need to prevent loss of events in case the endpoint returns an HTTP 400 response.

Which action should you perform?

A. Implement optimistic batching.
B. Implement asynchronous handshake validation.
C. Configure a dead-letter destination.
D. Configure a retry policy.

Answer: C
Explanation:
Configuring a dead-letter destination is exactly the right step to take.

By default, when an endpoint returns an HTTP 400 (Bad Request) or HTTP 413 (Payload Too Large) response, Azure Event Grid immediately stops delivery attempts and drops the message. It treats these specific errors as non-transient, client-side issues, meaning it skips its standard 24-hour retry policy.

Configuring a dead-letter destination ensures that these dropped events are safely preserved for future troubleshooting and reprocessing.

Reference:
https://turbo360.com/blog/azure-event-grid-dead-letter-monitoring

QUESTION 6
You are developing an AI API deployed to ACA. The API requires database credentials that are stored in Key Vault. Key Vault is configured to use Azure RBAC for access control.

The database credentials are rotated periodically by the security team. The application must always use the latest version of each credential without being redeployed and without exposing secrets in code or configurations.

You need to implement a secure secret access strategy that prevents credential exposure and fetches the latest version of each secret at runtime without redeploying the container.

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

NOTE: Each correct selection is worth one point.

A. Configure a Key Vault RBAC role assignment.
B. Configure a Key Vault access policy.
C. Export the secret during deployment.
D. Assign a system-assigned managed identity.
E. Retrieve the secret at runtime by using the SDK.

Answer: ADE
Explanation:
Step-by-Step Implementation Guide
1. Assign a System-Assigned Managed Identity

2. Configure Key Vault RBAC Role Assignment
Role Selection: Assign the Key Vault Secrets User role to the container app's managed identity.

Scope Limitation: Limit the scope of this assignment to the specific Key Vault or individual secrets rather than the entire resource group.

3. Retrieve the Secret at Runtime Using the SDK

Reference:
https://oneuptime.com/blog/post/2026-02-16-how-to-use-managed-identity-with-azure-container-apps-to-access-azure-services/view

QUESTION 7
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 on the review screen.

You plan to deploy a container to an Azure App Service API app named api1. You host the source code for api1 in a GitHub repository. The container uses the API key at runtime to connect to a backend service.

The container must be able to retrieve the API key at runtime without exposing it in the source repository or Git commit history.

You need to ensure that the API key remains outside of Git commit history and is available to the container at runtime.

Solution: Store the API key as a GitHub repository secret.

Does the solution meet the goal?

A. Yes

B. No

Answer: B
Explanation:
Correct:
* Store the API key in Azure Key Vault and reference it from an App Service application setting.

Storing the API key in Azure Key Vault and referencing it via App Service application settings is the recommended, secure approach. This strategy completely removes sensitive credentials from your GitHub repository and Git commit history while injecting them safely into your container environment at runtime.

Incorrect:
* Embed the API key as a hardcoded environment variable in the Dockerfile.
* Store the API key as a GitHub repository secret.

Reference:
https://www.qservicesit.com/full-stack-applications-on-azure

QUESTION 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 on the review screen.

You plan to deploy a container to an Azure App Service API app named api1. You host the source code for api1 in a GitHub repository. The container uses the API key at runtime to connect to a backend service.

The container must be able to retrieve the API key at runtime without exposing it in the source repository or Git commit history.

You need to ensure that the API key remains outside of Git commit history and is available to the container at runtime.

Solution: Embed the API key as a hardcoded environment variable in the Dockerfile.

Does the solution meet the goal?

A. Yes
B. No

Answer: B
Explanation:
Correct:
* Store the API key in Azure Key Vault and reference it from an App Service application setting.

Storing the API key in Azure Key Vault and referencing it via App Service application settings is the recommended, secure approach. This strategy completely removes sensitive credentials from your GitHub repository and Git commit history while injecting them safely into your container environment at runtime.

Incorrect:
* Embed the API key as a hardcoded environment variable in the Dockerfile.
* Store the API key as a GitHub repository secret.

Reference:
https://www.qservicesit.com/full-stack-applications-on-azure

QUESTION 9
You deploy multiple instances of a change feed processor to handle a high ingestion rate within Azure Cosmos DB for NoSQL.

Each processor instance must process a different subset of partitions.

You need to ensure the workload is load-balanced across all processor instances.

What should you configure?

A. indexing precision
B. strong consistency
C. lease container
D. autoscale throughput

Answer: C
Explanation:
To handle a high ingestion rate in Azure Cosmos DB for NoSQL, you must deploy multiple instances of the change feed processor using the same workflow name and same lease container.

Reference:
https://github.com/Azure/azure-cosmos-dotnet-v3/wiki/Comprehensive-Test-Suite-for-Azure-Cosmos-DB-Change-Feed-Processor:-All-Versions-and-Deletes

QUESTION 10
You are developing an AI search API that caches semantic search results in Redis.

Search results must remain cached for 10 minutes. If the underlying data changes, cached entries must NOT be returned.

You need to implement a cache-aside strategy to ensure data consistency.

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

NOTE: Each correct selection is worth one point.

A. Configure a cache notification for key space events.
B. Delete related cache keys when the source data changes.
C. Implement sliding expiration based on key access.
D. Configure a 10-minute Time to Live on each key.

Answer: BD
Explanation:
Delete keys on mutation. Implement an event-driven or database-hook system to purge related

Redis keys the moment the source data updates.

Use absolute expiration. Set a strict, un-extendable Time-To-Live (TTL) of 10 minutes when writing to the cache, ensuring the data naturally expires even if a deletion event is missed.

Incorrect:
[Not C]
Implementing a sliding expiration is not a good step for this specific architecture because it violates the requirement that entries must not be returned if the underlying data changes.

Reference:
https://medium.com/real-world-net/net-caching-strategies-compared-redis-vs-memory-vs-hybrid-b8b8be1e708d

QUESTION 11
You deploy an AI application across multiple Azure regions.

The application must be able to view writes across all regions within a predictable time window.

You need to determine the appropriate consistency level.

What are two consistency levels you can use to achieve the goal? Each correct answer presents a complete solution.

NOTE: Each correct selection is worth one point.

A. Session
B. Strong
C. Bounded staleness
D. Eventual

Answer: BC
Explanation:
The two appropriate consistency levels for a globally distributed AI application needing to view writes within a predictable time window are Bounded Staleness and Strong.

Bounded Staleness: This provides a guarantee that reads will lag behind writes by a user-specified "staleness window" (either in time or number of operations). It delivers high availability and predictable read consistency across all regions, balancing performance with strict read-order guarantees.

Strong: This provides linearizable and absolute consistency, ensuring that writes are seen synchronously across all regions. It features an exact zero-second predictable time window, though it comes at the expense of higher latency for multi-region operations due to the time required to replicate globally.

Reference:
https://learn.microsoft.com/en-us/azure/cosmos-db/consistency-levels

Thank You for Trying Our Product

Lead2pass Certification Exam Features:

★ More than **99,900** Satisfied Customers Worldwide.

★ Average **99.9%** Success Rate.

★ **Free Update** to match latest and real exam scenarios.

★ **Instant Download** Access! No Setup required.

★ Questions & Answers are downloadable in **PDF** format and **VCE** test engine format.

★ Multi-Platform capabilities - **Windows, Laptop, Mac, Android, iPhone, iPod, iPad**.

★ **100%** Guaranteed Success or **100%** Money Back Guarantee.

★ **Fast**, helpful support **24x7**.

View list of all certification exams: http://www.lead2pass.com/all-products.html

10% Discount Coupon Code: **ASTR14**