
AI-200

Developing AI Cloud Solutions on Azure

AZURE AI CLOUD DEVELOPER ASSOCIATE

FULL QUESTION & ANSWER SET

67

QUESTIONS & ANSWERS

Microsoft

VENDOR

Exam Syllabus & Objectives

About This Exam

The AI-200 certification validates your ability to design, build, and deploy intelligent cloud solutions on the Microsoft Azure platform. This exam tests practical expertise in containerizing applications, integrating AI workloads with managed data services, implementing event-driven architectures, and securing enterprise-grade solutions on Azure. Candidates must demonstrate proficiency across multiple Azure services and their integration patterns for AI and machine learning scenarios.

Core Knowledge Domains

Containerized Solutions on Azure (20-25%)

This domain covers the full lifecycle of container development and deployment. You will need to work with Azure Container Registry to build, store, version, and manage container images efficiently. Understanding how to use Azure Container Registry Tasks for automated image builds and how to push containers to registries is essential. You must also know how to deploy container workloads to Azure App Service with proper environment variable and secret injection. For orchestrated environments, you should be comfortable deploying applications to Azure Container Apps, configuring their environments, and managing revisions. Knowledge of event-driven scaling patterns using KEDA (Kubernetes Event-driven Autoscaling) within Container Apps is required. Additionally, you need hands-on skills with Azure Kubernetes Service (AKS), including deploying applications via manifest files, monitoring cluster health through logs and events, and troubleshooting connectivity issues across distributed containers.

AI Solutions Using Azure Data Management (25-30%)

This is the largest domain and focuses on storing, querying, and retrieving AI-related data. With Azure Cosmos DB for NoSQL, you must connect using SDKs, execute queries efficiently, and optimize performance through indexing policies and consistency level tuning. Vector database capabilities are critical—you should understand how to store embeddings, perform vector similarity searches for semantic retrieval, and implement change feed processors to track document changes. For Azure Database for PostgreSQL, competency includes SDK-based connections, schema design, and appropriate data type selection. Vector support is increasingly important; you need to work with pgvector extensions, optimize indexing for vector workloads, configure resources (compute, memory, storage) to handle vector operations, and build retrieval-augmented generation (RAG) patterns with metadata filtering. Azure Managed Redis knowledge includes caching strategies, expiration policies, cache invalidation, and using Redis for vector indexing to enable similarity searches.

Azure Service Integration and Messaging (20-25%)

This domain emphasizes connecting applications to Azure services through event and message-based architectures. You must understand how to use Azure Service Bus for queueing and back-end processing, including handling dead-letter queues, managing topics and subscriptions, and routing complex message workflows. Azure Event Grid is another critical component—you should know how to create event-driven workflows, apply filtering rules, publish custom events, and configure retry policies. Azure Functions rounds out this domain; you need to create serverless APIs with appropriate triggers (HTTP, timer, queue, etc.) and bindings that connect to other Azure services. Deploying and configuring function apps for scalability and reliability is expected.

Security, Monitoring, and Troubleshooting (20-25%)

Security practices are woven throughout this domain. Azure Key Vault is essential for storing secrets securely and

implementing rotation strategies. Azure App Configuration enables you to manage application settings and feature flags separately from code. For monitoring and troubleshooting, you must use OpenTelemetry SDKs to instrument applications and trace requests across distributed systems. Kusto Query Language (KQL) proficiency is required to write queries against Azure Monitor logs and metrics, enabling you to diagnose performance issues, identify errors, and gain insights into application behavior.

How This Exam Is Structured

The exam emphasizes practical, hands-on knowledge of Azure services working together in real-world AI and application scenarios. You should expect scenario-based questions and case studies that require you to design solutions across multiple services. The weighting indicates that data management and AI-specific features (such as vector databases and RAG patterns) represent the largest knowledge area, followed by containerization and secure, observable architectures. Success requires not just knowledge of individual services, but understanding how they integrate to form complete, production-ready solutions.

Question 1

SINGLE CHOICE

You are developing an AI application that processes customer embeddings stored in Azure Cosmos DB for NoSQL. Your application performs vector similarity searches frequently, but query latency is increasing as your dataset grows. You notice that Request Unit (RU) consumption is also higher than expected. Which combination of optimizations would most effectively reduce both query latency and RU consumption?

- A. Increase the provisioned throughput and implement a composite index that includes the embedding vector field and common filter attributes
- B. Implement a composite indexing policy that covers filter attributes and query predicates, and set the consistency level to eventual consistency for read operations
- C. Remove all indexes except the vector index to reduce write latency, and increase the provisioned throughput by 50%
- D. Implement full-text indexing on all fields and switch to strong consistency to ensure accuracy of vector similarity results

Answer:

B

Explanation:

The correct answer addresses both latency and RU consumption through two mechanisms: (1) composite indexes that cover commonly filtered attributes reduce the number of documents scanned during queries, and (2) eventual consistency for read-heavy workloads reduces RU cost per query. Simply increasing throughput (option A) solves the symptom but not the underlying inefficiency. Removing indexes (option C) would actually increase RU consumption by requiring full table scans. Strong consistency (option D) increases RU cost, not decreases it.

Question 2

SINGLE CHOICE

Your team is deploying a containerized AI inference service to Azure Kubernetes Service (AKS). The service must scale based on HTTP request volume and also respond to custom Kubernetes events generated by your batch processing system. You need to monitor the deployment and ensure you can quickly diagnose issues in the pod networking and inter-service communication. What approach should you use for scaling and monitoring?

- A. Use Kubernetes Horizontal Pod Autoscaler (HPA) for HTTP metrics and implement KEDA with custom event scalers; use kubectl logs and Events API for diagnostics
- B. Use Azure Container Apps with built-in autoscaling and Azure Monitor Application Insights for all diagnostics and event handling
- C. Configure Azure Load Balancer to manage scaling based on CPU, and use Azure Service Bus for all event routing and diagnostics
- D. Use KEDA for all scaling scenarios and configure Azure Application Gateway to monitor pod health and connectivity issues

Answer:

A

Explanation:

The correct answer correctly identifies that AKS deployments use Kubernetes HPA for metric-based scaling and KEDA for event-driven scaling, with native Kubernetes tools (kubectl logs, Events API) for troubleshooting pod connectivity and inter-service communication. Option B conflates Container Apps (a different Azure service) with AKS. Option C misrepresents the roles of Azure Load Balancer and Service Bus for AKS diagnostics. Option D incorrectly suggests Application Gateway is the primary diagnostic tool for AKS pod connectivity issues.

Question 3

SINGLE CHOICE

You are implementing a retrieval-augmented generation (RAG) system using Azure Database for PostgreSQL with pgvector. Your application must quickly retrieve relevant documents based on semantic similarity while filtering by metadata (e.g., date range, category). Vector similarity searches are timing out and consuming excessive compute. What configuration adjustments should you prioritize?

- A. Create a HNSW (Hierarchical Navigable Small World) index on the embedding column and apply metadata filtering before the vector search to reduce the search space
- B. Increase the shared_buffers parameter and use L2 distance instead of cosine similarity to speed up vector operations
- C. Store embeddings in Azure Managed Redis instead and query PostgreSQL only for metadata, then perform vector search in Redis
- D. Disable all other indexes on the metadata columns and perform full table scans to ensure optimal vector search performance

Answer:

A

Explanation:

The correct answer identifies the most effective optimization strategy: HNSW indexing dramatically accelerates vector similarity search, and metadata filtering reduces the working set before the expensive vector operation executes. Option B is incorrect because `shared_buffers` tuning alone won't solve the latency issue, and distance metric choice is a secondary consideration. Option C misunderstands the architecture—Managed Redis is useful for caching but shouldn't replace PostgreSQL for the RAG pattern. Option D would degrade performance for metadata filtering and defeat the purpose of indexing strategy optimization.

Question 4

SINGLE CHOICE

Your team is building an asynchronous AI processing pipeline using Azure Service Bus and Azure Functions. Long-running inference tasks are submitted as messages to a topic, but some tasks fail intermittently due to transient AI service timeouts. Currently, failed messages are being lost. You need to ensure failed messages are captured, investigated, and can be reprocessed. Which approach implements this requirement?

- A.** Configure the Service Bus topic subscription with a dead-letter queue (DLQ), set an appropriate max delivery count, and implement a separate Azure Function triggered by the DLQ to log and alert on failures
- B.** Implement message retry logic within the Azure Function using exponential backoff, then send failed messages to Azure Event Grid for routing to a storage account
- C.** Use Azure Service Bus message sessions and configure the Function to automatically re-queue failed messages to the main topic without capturing failure details
- D.** Store all messages in Azure Blob Storage first, then process them with Durable Functions to handle failures natively without using Service Bus DLQ

Answer:

A

Explanation:

The correct answer identifies the proper Service Bus pattern: configuring a DLQ with a max delivery count threshold ensures failed messages (after retries) are automatically moved to the DLQ, where a separate function can process them for logging, alerting, and reprocessing. Option B conflates Event Grid with Service Bus failure handling; Event Grid is for event distribution, not dead-letter processing. Option C avoids the core requirement of capturing and investigating failures. Option D unnecessarily decouples the message queue from the storage layer and adds complexity without leveraging Service Bus's built-in failure handling mechanisms.

Question 5

SINGLE CHOICE

You are securing a containerized AI application deployed to Azure App Service that requires access to multiple secrets (API keys, database connection strings) and configuration values. Some values change frequently, while secrets must rotate regularly. You need a solution that centralizes secret and configuration management and allows your application to retrieve updates without redeployment. What is the best approach?

- A.** Store secrets in Azure Key Vault and configuration in Azure App Configuration; use managed identities for authentication and reference both services from the App Service environment variables
- B.** Embed all secrets and configuration in a Docker image stored in Azure Container Registry with access control through RBAC
- C.** Store secrets in Azure App Service application settings and use a scheduled Azure Function to update configuration values daily
- D.** Use Azure Storage accounts with shared access signatures (SAS) for all configuration and secret storage, accessed via a custom middleware in the application

Answer:

A

Explanation:

The correct answer leverages the purpose-built Azure services: Key Vault for secrets with rotation support and audit logging, App Configuration for centralized settings management with dynamic updates, and managed identities to avoid storing credentials in the application. Option B embeds secrets in images, violating security best practices and preventing secret rotation without redeployment. Option C uses App Service settings (static) and manual updates, lacking dynamic configuration capability and proper secret isolation. Option D misuses Storage accounts (not designed for secret management) and SAS tokens (credentials that must still be rotated), and adds unnecessary middleware complexity.

Question 6

SINGLE CHOICE

You are monitoring a distributed AI application spanning Azure Functions, Azure Cosmos DB, and an AKS cluster running inference services. You need to trace requests end-to-end across services to diagnose latency bottlenecks and correlate logs across different components. Which monitoring approach best implements this requirement?

- A.** Instrument all services with OpenTelemetry SDKs to emit traces and metrics to Application Insights, then write KQL queries to correlate traces by operation ID across services
- B.** Enable diagnostic logging on each Azure service individually and manually correlate log entries by timestamp in Log Analytics
- C.** Deploy Azure Monitor Agent on all VMs and use built-in service maps to automatically visualize all dependencies without code instrumentation
- D.** Use Azure Application Insights only for Azure Functions, and configure separate Event Grid events for each AKS pod event to track execution flow

Answer:

A

Explanation:

The correct answer identifies the modern distributed tracing approach: OpenTelemetry instrumentation provides structured traces with operation IDs that correlate across all services, and Application Insights with KQL queries enables root cause analysis of latency. Option B relies on timestamps and manual correlation, which is error-prone at scale and doesn't capture request causality. Option C conflates Azure Monitor Agent (primarily for infrastructure metrics) with application-level distributed tracing. Option D incorrectly separates monitoring by service type and uses Event Grid (designed for event-driven workflows) inappropriately for execution flow tracking.

CASE STUDY 1**Background**

Proseware Inc. develops AI-powered knowledge management solutions for enterprise customers. The company is modernizing its platform to support semantic search, intelligent document retrieval, and real-time partner integrations.

The engineering team uses Python and Azure SDKs. The architecture is being redesigned to support containerized microservices, vector search workloads, and serverless backend processing.

Planned Application Architecture

Microservices are containerized by using Docker.

Code for containerized microservices and Azure Function apps is developed locally but stored in a GitHub repository.

Custom images for containerized microservices are stored in Azure Container Registry (ACR).

Base images are stored in Docker Hub. Custom images must be rebuilt automatically whenever their base images are updated.

Azure Cosmos DB for NoSQL stores documents, metadata, and vector embeddings.

Azure Functions generate vector embeddings of Azure Cosmos DB for NoSQL-hosted documents and send messages to Service Bus to trigger search index updates.

Azure Container Apps (ACA) apps host backend API services that provide semantic search across Azure Cosmos DB for NoSQL documents. API services process Service Bus messages and update search indexes.

Azure Kubernetes Service (AKS) processes batch vector embedding regeneration for existing Azure Cosmos DB for NoSQL documents (whenever the embedding model is changed).

An extranet-facing containerized webhook allows business partners to submit documents to be processed by internal AI workflows for semantic search and retrieval.

Monitoring

Telemetry generated by Azure resources is sent to Azure Monitor.

A Log Analytics workspace is used to collect ACA apps logs, AKS container logs, and Azure Functions apps logs.

Monitoring of Azure Functions is currently implemented by using Azure Application Insights SDK instrumentation.

Business Requirements

Embeddings for new or updated Azure Cosmos DB for NoSQL--hosted documents must be automatically generated.

Backend API services must scale automatically during business hours.

Cold start delay of backend APIs must be minimized.

Secrets must be stored outside of container images.

Developers must be able to correlate telemetry across Azure Functions hosts and apps.

All tracing must be implemented by using OpenTelemetry SDK instrumentation.

Development efforts must be minimized.

Technical Requirements

Container images must be built automatically and validated before code updates are merged into the main branch.

Image build automation must run inside the Azure Container Registry, eliminating dependency on local developer machines and external build services.

Dependency of image builds on local developer machines must be eliminated.

Event-driven scaling in ACA must occur based on the number of pending messages in the Azure Service Bus queue.

Azure Cosmos DB for NoSQL RU consumption must be minimized.

Vector similarity search must use embeddings stored in Azure Cosmos DB for NoSQL.

The partner-facing containerized webhook service must run on Azure App Service.

Secrets must NOT be stored in container images, source control, or application configuration directly. They must be accessed securely at runtime.

All secrets must be stored centrally in Azure Key Vault and accessed at runtime through a managed identity.

Azure App Service must supply secrets at runtime without relying on external services.

Resources and workloads must be deployed by using Bicep templates through an automated, version-controlled pipeline. Local and command-line deployments must be eliminated to ensure repeatable, auditable deployments.

Known issues

RU consumption spikes during vector similarity queries.

Question 7

DRAG DROP

You need to implement trace correlation according to the business requirements.

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.

NOTE: More than one order of answer choices is correct. You will receive credit for any of the correct orders you select.

QUESTION IMAGE:

Actions

- Redeploy the instrumented services.
- Configure a trace exporter in the OpenTelemetry SDK.
- Instrument the application code by using the OpenTelemetry SDK.
- Call `TelemetryClient.TrackEvent()` within service methods.
- Implement a view in the OpenTelemetry SDK.

Answer Area

ANSWER: ITEMS PLACED IN THE CORRECT POSITION

Actions

- Redeploy the instrumented services.
- Configure a trace exporter in the OpenTelemetry SDK.
- Instrument the application code by using the OpenTelemetry SDK.
- Call `TelemetryClient.TrackEvent()` within service methods.
- Implement a view in the OpenTelemetry SDK.

Answer Area

- Instrument the application code by using the OpenTelemetry SDK.
- Configure a trace exporter in the OpenTelemetry SDK.
- Redeploy the instrumented services.

Question 8

SINGLE CHOICE

You need to deploy Azure function resources and apps by using an automated, version-controlled failed pipeline that supports declarative infrastructure deployment. What should you use?

A. Local Git deployment

B. GitHub Actions

C. Azure Functions Core Tools

D. Azure CLI

Answer:

B

Explanation:

To deploy Azure Function resources and apps using an automated, version-controlled pipeline with declarative infrastructure deployment, you should use Infrastructure as Code (IaC) tools:

- **Azure Resource Manager (ARM) templates** - Microsoft's native declarative template language for Azure resources
- **Bicep** - A modern, more readable ARM template language developed by Microsoft
- **Terraform** - A popular multi-cloud IaC tool that supports Azure with version control and state management

These tools support version control, declarative syntax, and can be integrated into CI/CD pipelines for automated deployment.

Question 9

MULTIPLE CHOICE

You need to address the known issue resulting from vector similarity queries.

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

NOTE: Each correct selection is worth one point.

A. Add a composite index on the vector fields and metadata properties of the container.

B. Modify the indexing precision of the vector fields.

C. Set the account consistency level to Strong.

D. Change the vector index type from flat to quantizedFlat or diskANN.

Answer:

B, D

Explanation:

Note: The question references a 'known issue resulting from vector similarity queries' but does not provide the options. Vector similarity query issues typically require:

- Creating appropriate vector indexes (pgvector extension in PostgreSQL)
- Configuring proper distance metrics and index types
- Tuning query parameters for performance
- Potentially enabling specific extensions or features

Without the available options, specific actions cannot be determined. Please provide the answer choices to select the correct solutions.

Question 10

HOTSPOT

You need to configure vector embedding updates according to the business and technical requirements.

Which information should you use? To answer, select the appropriate option in the answer area.

NOTE: Each correct select is worth one point.

QUESTION IMAGE:

Update vector embeddings

Requirement

Identify document changes that should trigger vectorization.
Scale out the vectorization processing.

Scale out the vectorization processing.

Configuration

Change feed processor
Periodic full container scan
Cross-partition SELECT query

Lease container

Strong consistency level
RU throughput on the container

ANSWER: CORRECT AREA HIGHLIGHTED

Update vector embeddings

Requirement

Identify document changes that should trigger vectorization.
Scale out the vectorization processing.

Scale out the vectorization processing.

Configuration

Change feed processor
Periodic full container scan
Cross-partition SELECT query

Lease container

Strong consistency level
RU throughput on the container

Question 11

SINGLE CHOICE

You need to configure a connection string for the partner-facing service according to the technical requirements.

What should you use?

- A.** Azure Key Vault references in App Service settings
- B.** GitHub secrets
- C.** Azure Container Registry Helm chart package
- D.** Dockerfile ENV instructions

Answer:

A

Explanation:

The question mentions configuring a connection string for a 'partner-facing service' but does not provide the available options or context about what type of service this is.

Generally, connection strings for partner-facing services could use:

- Azure Key Vault for secrets management
- Azure App Configuration for configuration management
- Connection string directly in environment variables (less secure)
- Managed Identity authentication (passwordless)

Please provide the answer choices to determine the most appropriate solution for your specific scenario.

Question 12

SINGLE CHOICE

You are developing an AI application that processes customer feedback using Azure Container Apps. The application must automatically scale based on the number of messages in an Azure Service Bus queue. Currently, the app scales only based on CPU and memory metrics. You need to implement event-driven autoscaling without modifying your application code.

What should you do?

- A.** Configure Kubernetes Event-driven Autoscaling (KEDA) with the Azure Service Bus scaler in Container Apps

- B. Implement custom metrics collection and push them to Azure Monitor for Container Apps to consume
- C. Deploy the application to Azure Kubernetes Service (AKS) instead, as Container Apps does not support event-driven scaling
- D. Use Azure Functions with Service Bus triggers to manage scaling logic independently

Answer:

A

Explanation:

The correct answer is to configure KEDA with the Azure Service Bus scaler in Container Apps. KEDA is the recommended way to implement event-driven autoscaling in Container Apps, and it includes built-in scalers for Azure Service Bus that can scale based on queue depth without application code changes.

The second option is incorrect because while Azure Monitor can collect metrics, it does not provide automatic scaling based on custom metrics in Container Apps. The third option is wrong because Container Apps fully supports KEDA-based event-driven autoscaling. The fourth option misses the point—Azure Functions are not needed when Container Apps already has KEDA integration.

Question 13

SINGLE CHOICE

You are developing a Java application to be deployed in Azure. The application stores sensitive data in Azure Cosmos DB.

You need to configure Always Encrypted to encrypt the sensitive data inside the application.

What should you do first?

- A. Create a Microsoft Entra ID managed identity and assign the identity to a new Azure Key Vault instance.
- B. Create a new container to include an encryption policy with the JSON properties to be encrypted.
- C. Create a customer-managed key (CMK) and store the key in a new Azure Key Vault instance.
- D. Create a data encryption key (DEK) by using the Azure Cosmos DB SDK and store the key in Azure Cosmos DB.