DUMPSBOSS.

Developing AI Apps and Agents on Azure

Microsoft AI-103

Version Demo

Total Demo Questions: 7

Total Premium Questions: 71

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Implement generative AI solutions	13
Topic 2, Implement agentic solutions	39
Topic 3, Implement computer vision solutions	1
Topic 4, Implement natural language processing solutions	3
Topic 5, Implement knowledge mining and information extraction solutions	10
Topic 6, Case Study Contoso, Ltd Overview	5
Total	71

QUESTION NO: 1

You need to configure an indexing pipeline for Agent1 to retrieve the relevant product information in storage1. The solution must

meet the technical requirement.

Which two built-in skills should you use? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Language Detection

B. Entity Recognition

C. Merge

D. Azure OpenAI Embedding

E. Text Split

F. key phrase extraction

ANSWER: D E

Explanation:

The correct built-in skills are Azure OpenAI Embedding and Text Split. In an Azure AI Search indexing pipeline that supports retrieval-augmented generation, source documents such as product information in Blob Storage typically need to be divided into smaller passages and then converted into vectors. Text Split is the built-in skill designed to chunk large text into smaller units, which improves retrieval granularity and helps keep each chunk within embedding model token limits. Azure OpenAI Embedding is the built-in skill used during indexing to generate vector embeddings for those chunks by calling an Azure OpenAI embedding model deployment.

Together, these skills implement the core integrated vectorization pattern in Azure AI Search: chunk the content, embed each chunk, and store the resulting vectors in a searchable index so an agent can retrieve semantically relevant product information for user queries. Microsoft's guidance for integrated vectorization specifically describes using a skillset with Text Split for chunking and Azure OpenAI Embedding for vector generation during indexing. See Integrated vectorization in Azure AI Search and the Azure OpenAI Embedding skill documentation.

QUESTION NO: 2

You are creating an image-editing workflow in a Microsoft Foundry project. The workflow must meet the following requirements:

Ensure that background objects can be removed by applying a mask-based inpainting edit. Preserve the original lighting and style of the edited images.

Use the built-in image editing controls, NOT a custom model.

You need to ensure that image edits apply exclusively inside the masked area. How should you configure the workflow?

A. Enable text_to_image mode and a prompt describing the desired background removal.

B. Set generation mode to image_variation and provide the original image as a reference.

C. Enable image_to_image mode and a high-strength value to regenerate the full image based on the prompt.

D. Enable mask_inpainting and supply both the input image and a mask indicating which part of the image to modify.

ANSWER: D

Explanation:

Enable mask_inpainting and supply both the input image and a mask indicating which part of the image to modify. is correct because mask-based inpainting is the built-in image editing pattern intended for changing only a selected region of an existing image. In Azure OpenAI image editing workflows available through Azure AI Foundry, the edit request can include the original image, an instruction prompt, and a mask. The mask identifies the pixels that are eligible for modification, allowing the model to remove or replace background objects while keeping the unmasked portions of the image intact. This directly supports the requirement to preserve the original lighting, style, and composition outside the selected area because the workflow is not asking the model to recreate the whole image from scratch; it is constraining the edit to the masked region.

Microsoft's image generation and editing guidance describes using image edit capabilities with an input image and mask for targeted modifications, and Azure AI Foundry provides these capabilities as managed model features rather than requiring a custom model. See Microsoft's guidance for Azure OpenAI image generation and editing in Generate and edit images with Azure OpenAI and the Azure AI Foundry overview in What is Azure AI Foundry?.

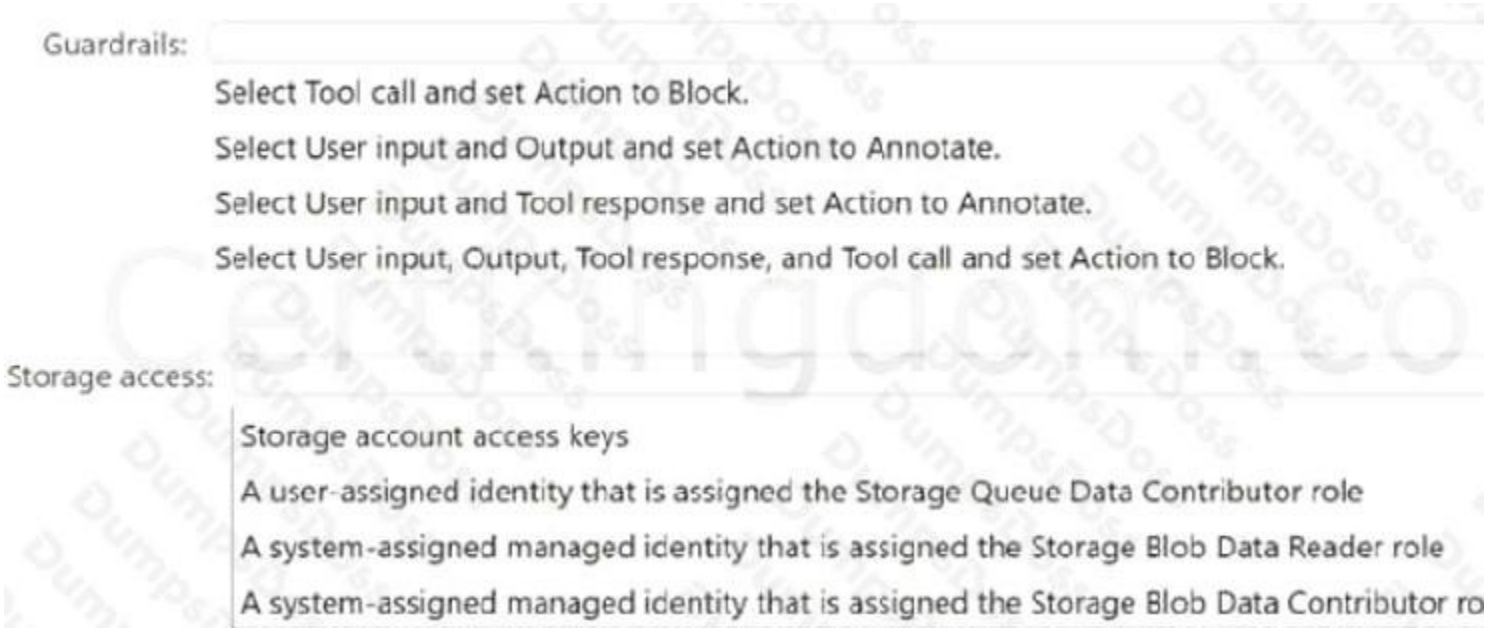
QUESTION NO: 3 - (HOTSPOT)

You have a Microsoft Foundry project that contains a customer support agent built by using the Foundry Agent Service. The agent uploads user-provided screenshots to Azure Storage through a ticketing tool and receives a blob URL for additional reasoning.

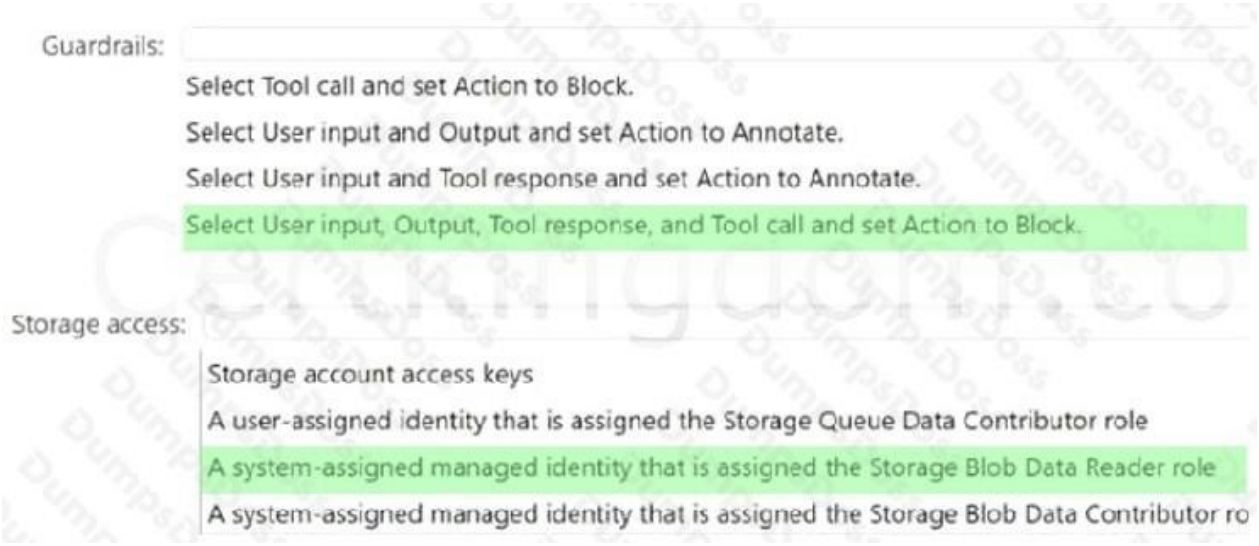
You need to use image moderation during agent runs and prevent harmful content from being returned during runs. Azure AI

Content Safety must access the images by using the blob URL. The solution must follow the principle of least privilege. What should you configure for Content Safety? To answer, select the appropriate options in the answer area.

NOTE: Each correct selection is worth one point.



ANSWER:



Explanation:

The correct configuration is to apply blocking guardrails across the full agent run path and to let Azure AI Content Safety read the screenshots from Blob Storage by using a managed identity with only read permissions. In this scenario, the screenshot may move through several agent intervention points: it starts with the user interaction, is handled by a tool, is returned from the tool as a blob URL, and can influence the final response. Because the requirement is to prevent harmful content from being returned during runs, the guardrail action needs to be set to Block rather than only adding annotations. Applying the guardrail to User input, Output, Tool response, and Tool call ensures moderation is enforced wherever the image reference or related harmful content may appear in the agent workflow. Microsoft describes agent guardrails as controls that can be applied at multiple points in an agent run to help detect and intervene on unsafe content; see Azure AI Foundry safety and evaluations and Azure AI Foundry Agent Service concepts.

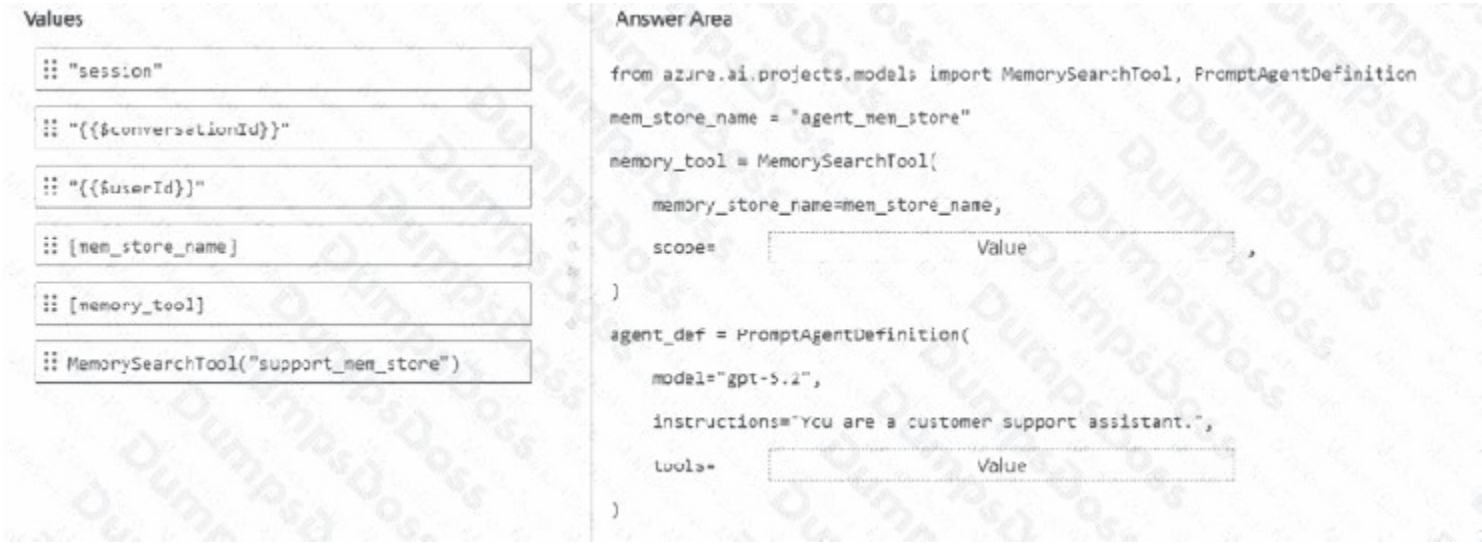
For the storage portion, Content Safety must access images by blob URL, so it only needs permission to read blob data. A system-assigned managed identity is appropriate because it is tied directly to the Azure AI Content Safety resource and avoids managing credentials or storage account keys. Assigning the Storage Blob Data Reader role at the container or storage account scope gives the Content Safety resource the data-plane read permission needed to retrieve the image while following least privilege. Microsoft documents managed identities as the recommended way for Azure services to access resources securely, and Azure Storage RBAC roles define Storage Blob Data Reader as the role for reading blob data; see managed identities for Azure resources and assign Azure roles for blob data access.

QUESTION NO: 4 - (DRAG DROP)

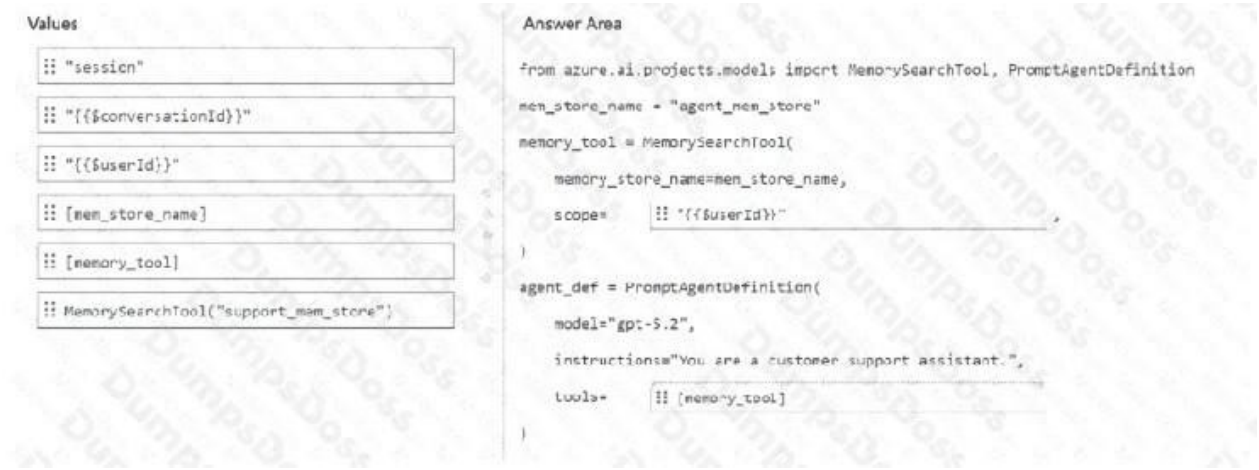
You have a Microsoft Foundry project that contains an agent.

You need to enable long-term memory to ensure that the agent can recall user preferences across separate conversations. Stored memories must be isolated per authenticated user without the client application manually generating user IDs.

How should you complete the Python code? To answer, drag the appropriate values to the correct targets. Each value may be used once, more than once, or not at all.



ANSWER:



Explanation:

The correct completion is to set the memory tool scope to **"{{userId}}"** and to attach the created memory tool to the agent by setting **tools=[memory_tool]**. The scope value is important because long-term memory in Azure AI Foundry Agent Service is partitioned by scope. When the scope uses the user identity template, the service can associate stored memories with the authenticated user context rather than with a single conversation or with an ID that the application has to invent and manage. This matches the requirement that memories persist across separate conversations while remaining isolated per authenticated user.

The **MemorySearchTool** object is created earlier in the code by referencing the existing memory store and the selected scope. That object is what gives the agent access to the configured long-term memory store, so the agent definition must include the same object in its **tools** collection. In Python, the tools parameter expects a list of tool definitions or tool objects, which is why the value should be **[memory_tool]**. With that configuration, the agent can use the memory store during future interactions to retrieve and update remembered user preferences in the correct user-specific partition.

This aligns with Microsoft's guidance for configuring memory tools in Azure AI Foundry agents, where memory is attached to an agent through a tool and scoped so stored information can be retrieved appropriately in later threads or conversations. For more detail, see Microsoft Learn on using memory tools with Azure AI Foundry Agent Service and the Azure AI Foundry Agents Python quickstart.

QUESTION NO: 5

You have a Microsoft Foundry project that contains a prompt agent used by a customer support web app. The agent is invoked from a Python service that does NOT run in the Foundry portal.

You need to implement end-to-end tracing to capture latency breakdowns and exceptions across agent runs. Which two components can you use? Each correct answer presents a complete solution.

NOTE: Each correct selection is worth one point.

A. a Log Analytics workspace

B. OpenTelemetry

C. Application Insights

D. the Azure Monitor Agent

E. Microsoft Sentinel

ANSWER: B C

Explanation:

OpenTelemetry and Application Insights are the correct components for implementing end-to-end tracing for an Azure AI Foundry prompt agent invoked from an external Python service. OpenTelemetry is the instrumentation standard used to collect and correlate distributed traces from application code, SDK calls, and agent execution. In this scenario, the Python service runs outside the Foundry portal, so instrumenting the client application with OpenTelemetry enables trace propagation and captures useful telemetry such as request timing, spans, exceptions, and dependency calls across the agent run.

Application Insights is the Azure Monitor-backed destination commonly used with Azure AI Foundry tracing. It stores and visualizes the collected telemetry so teams can investigate latency, failures, exceptions, and request flows. Azure AI Foundry documentation describes tracing with Application Insights and OpenTelemetry so developers can observe agent behavior and troubleshoot production applications beyond the portal experience. Together, these components provide the required distributed tracing pipeline: OpenTelemetry captures and exports the telemetry, while Application Insights provides the monitoring and analysis backend. See Trace your application with Azure AI Foundry and Enable Azure Monitor OpenTelemetry for applications.

QUESTION NO: 6

You have a Microsoft Foundry project named Project1 that contains an agent. The agent uses an OpenAPI 3.0 specification to call

an external weather service.

The weather service requires a key to be passed in an HTTP header. The key value is stored as a connection in Project1.

You need to ensure that the key value from the connection is included automatically whenever the OpenAPI tool is invoked.

What should you configure in the OpenAPI specification?

A. an Azure Key Vault connection

B. a header parameter defined for each operation

C. an API key security scheme

D. a Bearer token security scheme

ANSWER: C

Explanation:

an API key security scheme is correct because an OpenAPI tool should describe credential requirements through the OpenAPI security model, not by hard-coding or manually supplying the key as a normal operation input. In OpenAPI 3.0, an API key that must be sent in an HTTP header is represented under components.securitySchemes with type set to apiKey, in set to header, and name set to the required header name. The relevant operation, or the whole API, then references that

scheme through the security section. In Azure AI Foundry Agent Service, the OpenAPI tool can use a project connection that stores the secret value, and the declared API key security scheme tells the runtime how to apply that connection when invoking the external API. This matches the requirement to include the stored key automatically every time the tool is called. Microsoft's guidance for OpenAPI tools in Azure AI Foundry Agents describes using OpenAPI specifications with authentication schemes and project connections for secured APIs. See Use OpenAPI specified tools and Add connections in Azure AI Foundry.

QUESTION NO: 7 - (HOTSPOT)

You need to configure the model deployment for Agent1 to meet the technical requirements. What should you configure? To answer, select the appropriate options in the answer area.

NOTE: Each correct selection is worth one point.

ANSWER:

Explanation:

The correct configuration is to use the **Standard** deployment type and set the version update policy to **Once the current version expires**. A Standard deployment is the appropriate choice when the workload must stay tied to a specific Azure region and use pay-as-you-go capacity rather than reserved provisioned throughput. In the scenario, Agent1 must meet regional data-processing requirements in an EU Azure region and must support variable customer support demand without committing to provisioned capacity. Microsoft describes Standard deployments as regional deployments, while global deployment options are designed to route traffic across a broader set of infrastructure and provisioned deployments are

intended for reserved throughput capacity. See Microsoft's deployment type guidance here: Azure OpenAI deployment types.

The version update policy should be **Once the current version expires** because it balances production stability with service continuity. For an agent used in customer support, keeping the same model version during its supported lifecycle helps preserve predictable behavior and reduces the chance that responses change unexpectedly because a newer default model version became available. At the same time, allowing an update when the current version expires prevents the deployment from being left on a retired model version. Microsoft's model versioning documentation explains how deployment version policies control whether deployments update when a new default version is released or only when the current version reaches expiration. This makes the expiration-based policy the best fit for a stable production agent that still needs to remain supported over time. Reference: Azure OpenAI model versions.