

Microsoft Certified: Azure AI Cloud Developer Associate

Dominios, pesos oficiales y habilidades clave del temario de AI-200, para repasar antes del examen.

Nivel: Intermediate (Associate) Duración: 120 min Nota de corte: 700/1000 Fuente: guía de estudio oficial de Microsoft Learn

1. Develop containerized solutions on Azure

20–25%

Implement container application hosting

- Azure Container Registry guarda imágenes por repositorio, etiqueta y manifiesto; extraer por resumen fija la versión. Usa etiquetas únicas para desplegar y bloquea las desplegadas con `az acr repository update`. Reinsertar una etiqueta deja manifiestos huérfanos: elimínalos por resumen, con `acr purge` o con una directiva de retención. `az acr import` copia imágenes sin Docker.
- ACR Tasks compila imágenes en la nube: `az acr build` ejecuta una tarea rápida sin Docker local e inserta la imagen salvo con `--no-push`. `az acr task create` admite desencadenadores de confirmación, imagen base (con etiqueta estable) y temporizador cron en UTC. Las tareas de varios pasos se definen en YAML con pasos `build`, `push` y `cmd`, y acceden a otros recursos con identidades administradas.
- App Service ejecuta contenedores personalizados que extraen la imagen de Azure Container Registry, idealmente con identidad administrada (`AcrPull` y `acrUseManagedIdentityCreds`). Las opciones de aplicación llegan al contenedor como variables de entorno (`WEBSITES_PORT` si no escucha en el 80; `__` en lugar de dos puntos en Linux) y los secretos se resuelven con referencias `@Microsoft.KeyVault` mediante la identidad de la aplicación.

Implement container-orchestrated solutions

- En Azure Container Apps cada cambio en `properties.template` (imagen, sufijo, reglas de escalado) crea una revisión inmutable; los secretos, la entrada y la división de tráfico son cambios de ámbito de aplicación. El modo único cambia el tráfico sin interrupción cuando la nueva revisión está lista; el modo de varias revisiones añade pesos, etiquetas con URL propia y despliegues azul-verde. El entorno es el límite seguro de red y registros.
- Azure Container Apps escala con KEDA: las reglas HTTP y TCP usan solicitudes o conexiones simultáneas, y las personalizadas copian el type y los metadatos de un escalador de KEDA. La autenticación usa secretos (`trigger-Parameter` y `secretRef`) o una identidad administrada con la propiedad `identity`. Sin entrada ni regla personalizada, la aplicación escala a cero y no vuelve a arrancar; los trabajos por eventos inician una ejecución por evento.
- En AKS, `az aks get-credentials` configura `kubectl` y `kubectl apply -f` crea los objetos del manifiesto. Un `Deployment` solo crea una revisión al cambiar `.spec.template`; `maxSurge` y `maxUnavailable` controlan la actualización gradual, `Recreate` evita versiones simultáneas y `kubectl rollout undo` revierte. Los Service eligen pods por selector, el selector del `Deployment` es inmutable y los sondeos de preparación, inicio y actividad tienen usos distintos.
- En Azure Container Apps, la transmisión de registros (`az containerapp logs show` con `--type console` o `system`) muestra en directo `stdout/stderr` y los mensajes del servicio, mientras que Log Analytics guarda `ContainerAppConsoleLogs_CL`, `ContainerAppSystemLogs_CL` y, si se habilitan, `ContainerAppHTTPLogs`. En AKS se diagnostica con `kubectl describe`, `get events`, `logs --previous`, `debug` y `EndpointSlices`.

2. Develop AI solutions by using Azure data management services

30–35%

Develop AI solutions by using Azure Cosmos DB for NoSQL

- Con el SDK de Python se crea un único CosmosClient (con DefaultAzureCredential y un rol de plano de datos como Cosmos DB Built-in Data Contributor) y se obtienen la base de datos y el contenedor. read_item hace lecturas puntuales por id y clave de partición; query_items ejecuta consultas parametrizadas con @ y puede ser entre particiones. Las combinaciones solo funcionan dentro de un elemento.
- La directiva de indexación determina el coste: excluir rutas que no se consultan abarata las escrituras, los índices compuestos son necesarios para ORDER BY de varias propiedades y optimizan un filtro de rango cada uno, y el TTL requiere el modo consistent. Strong y Bounded staleness duplican el coste de las lecturas; Session garantiza leer las propias escrituras con lecturas de una réplica.
- Azure Cosmos DB for NoSQL guarda las incrustaciones junto a los datos. Requiere habilitar EnableNoSQLVectorSearch y crear el contenedor con una directiva vectorial (path, dataType, dimensions, distanceFunction) y un índice flat, quantizedFlat o diskANN. Las consultas usan VectorDistance con TOP N, y la búsqueda híbrida combina VectorDistance y FullTextScore con RRF en ORDER BY RANK.
- El procesador de fuente de cambios (SDK de .NET y Java) lee los cambios de un contenedor, guarda su progreso en un contenedor de concesiones y reparte las concesiones entre instancias con el mismo processorName y nombres de instancia distintos, con entrega de al menos una vez. En Python se usa el modelo de extracción con tokens de continuación o el desencadenador de Azure Functions; el modo de versión más reciente no captura eliminaciones.

Develop AI solutions by using Azure Database for PostgreSQL

- Desde Python se usa psycopg con un token de Microsoft Entra (ámbito ossrdbms-aad) como contraseña y sslmode=require, renovando el token porque caduca; las identidades administradas se dan de alta con pgaadauth_create_principal en la base de datos postgres. Para rendir bien conviene agrupar conexiones (psycopg_pool, Npgsql o PgBouncer en el puerto 6432), agrupar consultas, cargar con COPY y reintentar con retroceso exponencial.
- La extensión se llama vector y se crea en cada base de datos; las columnas vector(n) validan las dimensiones y son necesarias para indexar, halfvec reduce el tamaño y los índices de expresión permiten indexar por conversión. Los metadatos que más se filtran van en columnas tipadas con índices B-tree (columna inicial del compuesto), JSONB con GIN solo sirve para contención, y las particiones o tablas separadas aíslan inquilinos.
- HNSW da mejor equilibrio velocidad-exhaustividad y se crea sin datos; IVFFlat se crea tras cargar datos y se ajusta con ivfflat.probes, y HNSW con hns.ef_search. El índice solo se usa si el operador coincide con la clase de operadores y ORDER BY es una distancia ascendente con LIMIT. Para reducir cálculo: producto interno con vectores normalizados, recorridos iterativos al filtrar, cuantificación binaria y menos dimensiones.
- Las búsquedas vectoriales con pgvector consumen CPU para calcular distancias y memoria para mantener los índices en caché; el nivel Memory Optimized ofrece más memoria por vCore y el nivel Burstable no sirve para carga sostenida porque agota los créditos de CPU. El tamaño de proceso limita las IOPS utilizables, Premium SSD v2 permite ajustar IOPS y rendimiento por separado, y el almacenamiento solo se puede ampliar, nunca reducir.
- Con pgvector, la búsqueda semántica ordena por un operador de distancia (como <=>) y combina filtros WHERE, umbrales de distancia y LIMIT; con índices aproximados el filtro se aplica tras recorrer el índice, así que conviene activar escaneos iterativos o particionar por inquilino. En RAG, PostgreSQL es el recuperador: fragmentos enlazados a su documento, ventanas de contexto con fragmentos adyacentes y metadatos para citar.

Integrate Azure Managed Redis in AI solutions

- En Azure Managed Redis, cache-aside lee primero la caché, consulta el almacén si falla y guarda el resultado con un TTL (por ejemplo, SETEX); al escribir, se actualiza el almacén y después se invalida la clave. La directiva de expulsión predeterminada es volatile-lru, que solo expulsa claves con TTL; SET sin KEeptTL elimina el TTL, conviene borrar por patrón con SCAN en lugar de KEYS y añadir una pequeña variación aleatoria a los TTL.
- Azure Managed Redis hace búsqueda vectorial con el módulo Redisearch, que se habilita al crear la instancia y exige la directiva de clústeres Enterprise y NoEviction. FT.CREATE define el campo VECTOR (FLAT exacto o HNSW aproximado, con TYPE, DIM y DISTANCE_METRIC) y FT.SEARCH con DIALECT 2 lanza consultas KNN, por rango o filtradas por metadatos.

3. Connect to and consume Azure services

20–25%

Develop event- and message-based AI solutions

- Service Bus usa colas para entregas punto a punto y temas con suscripciones filtradas para publicar a muchos. En modo peek-lock el consumidor completa, abandona, aplaza o envía a mensajes fallidos cada mensaje; superar el número máximo de entregas o expirar con la opción activada lleva el mensaje a la subcola de mensajes fallidos, que se lee con sub_queue y no se limpia sola.
- Una suscripción de Event Grid filtra por tipo de evento, por inicio o final del asunto y con filtros avanzados (valores de un filtro con OR, filtros distintos con AND, sin distinguir mayúsculas). La entrega es al menos una vez y sin orden garantizado: Event Grid reintenta con espera exponencial hasta agotar el tiempo de vida o los intentos, lo que ocurra antes, y solo guarda mensajes fallidos si configuras un contenedor de blobs.

Develop and implement Azure Functions

- Cada función tiene exactamente un desencadenador y cero o más enlaces de entrada y salida. Una API sin servidor usa el desencadenador HTTP: la plantilla route define la URL (con el prefijo api, configurable en host.json), authLevel decide qué clave de acceso se exige y las expresiones de enlace como {id} o {Query.id} pasan datos de la solicitud a otros enlaces sin escribir código del SDK.
- Flex Consumption es el plan sin servidor recomendado: solo admite implementación de paquetes, se configura con functionAppConfig (memoria de instancia, instancias siempre listas, contenedor de implementación) y usa actualizaciones graduales en vez de espacios. En Premium y Dedicated, los espacios permiten intercambiar ensayo y producción, y las conexiones con identidad usan sufijos como __accountName o __fullyQualifiedNamespace.

4. Secure, monitor, and troubleshoot Azure solutions

20–25%

Implement secure Azure solutions

- Las aplicaciones leen secretos con SecretClient y DefaultAzureCredential (identidad administrada en Azure) o con referencias @Microsoft.KeyVault en la configuración de aplicación, y el rol Key Vault Secrets User basta para leer. set_secret crea una nueva versión; la rotación se automatiza con el evento SecretNearExpiry de Event Grid y una función, y con dos credenciales se evita el tiempo de inactividad.
- Azure App Configuration guarda pares clave-valor con etiquetas, marcas de características y referencias de Key Vault, que solo almacenan un URI: la aplicación lee el secreto con su propia identidad. El proveedor de Python (load) combina selectores, recorta prefijos y actualiza bajo demanda con refresh() y una clave centinela. Las instantáneas inmutables y Microsoft Entra ID con roles de datos completan un uso seguro.

Monitor and troubleshoot Azure solutions

- El seguimiento distribuido con OpenTelemetry correlaciona intervalos entre servicios mediante W3C Trace Context: el trace-id se convierte en operation_Id en Application Insights. En Python, configure_azure_monitor() lee la cadena de conexión, instrumenta bibliotecas comunes y admite muestreo, procesadores de intervalos y recursos (service.name define el rol en el mapa). Sin instrumentación, el contexto se propaga con inject y extract.
- KQL encadena operadores con barras verticales: where filtra pronto por TimeGenerated con ago(), summarize agrega por bin() y make-series rellena con ceros los intervalos vacíos para el análisis de series. has busca términos completos y es más rápido que contains; join usa innerunique por defecto y lookup sirve para tablas de dimensiones pequeñas. Con muestreo, cuenta con sum(ItemCount) en las tablas App* del área de trabajo.