

ACR Tasks triggers

Name the three ACR Tasks trigger types. Which events are enabled by default? What is NOT a trigger?

Answer on the reverse.

ACR pull and push roles

Which RBAC roles grant image pull and push? What changes on an ABAC-enabled registry?

Answer on the reverse.

ACR Tasks build types

Name the three ACR Tasks scenarios. What makes a task rebuild on every commit? What is NOT an ACR Task?

Answer on the reverse.

ACR tags and digests

Stable tag, unique tag or digest - which pins one exact image? How do you get an image into ACR securely?

Answer on the reverse.

ACR pull and push roles

- **Standard (RBAC Registry Permissions):** AcrPull to pull, AcrPush to push.
- **ABAC-enabled** (RBAC Registry + ABAC Repository Permissions): **Container Registry Repository Reader** to pull, **Container Registry Repository Writer** to push.
- On an ABAC registry the legacy AcrPull / AcrPush / AcrDelete roles are **not honored**. One word in the stem flips the answer.

ACR Tasks triggers

- **Source code update** - commit (enabled by default) and pull request (**DISABLED** by default). Needs a personal access token.
- **Base image update** - enabled by default. ACR-hosted base image rebuilds immediately; Docker Hub / MCR checked at a **random** 10-60 min interval. Task must have built once already.
- **Timer** - 5-field cron, down to once per minute.
- **Not triggers:** ACR webhooks (outbound notification) and Artifact Cache rules (mirror an upstream image).

ACR tags and digests

- **Stable tag** (1.0, latest, any reused tag) - keeps receiving updates. Use for **base images**, NOT for deployments.
- **Unique tag** (build ID, date stamp) - a new tag for every push. Use for **deployments**, so a restart or scale-out never pulls a newer image.
- **Manifest digest** - unique SHA-256 hash per image. Pull with repo@sha256:... to **guarantee** the version, even if the same tag is pushed again.
- **Push flow** - az acr login (Microsoft Entra token, valid **3 hours**) -> docker tag with the login server (myregistry.azurecr.io/repo:v1) -> docker push.
- No tag given -> **latest** by default. Lock a deployed tag by setting write-enabled to false.

ACR Tasks build types

- **Quick task** - az acr build: think docker build + docker push **in the cloud**. No local Docker Engine needed.
- **Automatically triggered task** - az acr task create with --context set to a GitHub or Azure DevOps repo (#branch), --file Dockerfile and --git-access-token (PAT). The **repo is the build context**.
- **Multi-step task** - a YAML file of build, push and cmd steps, run with az acr run.
- **Commit flow** - push code to the repo -> commit trigger queues the task -> image built in Azure and pushed to ACR. {{.Run.ID}} in the tag gives a unique tag per run.
- **Trap:** a GitHub Actions workflow or Azure DevOps pipeline also builds images, but it is NOT an ACR Task. Commit and PR triggers do NOT work with GitHub Enterprise repos.

App Service pull from ACR

Two ways App Service can authenticate to ACR.
Which one stores no credentials, and what role does it need?

Answer on the reverse.

App Service app settings

How does App Service pass configuration to a container? App setting, connection string or Dockerfile
- which fits?

Answer on the reverse.

App Service Key Vault refs

What does a Key Vault reference look like? What must the app have before it resolves? How fast is a rotated secret picked up?

Answer on the reverse.

KEDA scale rule

Scaler metadata key for Service Bus vs Storage queue? Polling interval, cool-down, formula, replica defaults?

Answer on the reverse.

App Service app settings

- **App settings** - name-value pairs injected as **environment variables** when the container starts. Same image, different values per environment. No extra service needed.
- **Encrypted at rest** - every app setting. Names: letters, numbers and underscores; on Linux use `__` in place of `:` for nested .NET keys.
- **Connection strings** - meant for databases; the env var gets a type prefix (SQLAZURECONNSTR_, CUSTOMCONNSTR_). Non-.NET apps should just use app settings.
- **Slot settings** - stick to the slot during a swap (environment names, endpoints, feature flags).
- **Trap:** a value hardcoded in the Dockerfile is baked into the image and committed with the code. App settings supply it at runtime instead.

App Service pull from ACR

- **Managed identity** (recommended) - system-assigned or user-assigned identity on the web app with **AcrPull** on the registry. Nothing to store or rotate.
- **System-assigned** - created and deleted with the app. **User-assigned** - separate resource, shareable across apps, can get AcrPull before the app exists.
- **Admin credentials** - registry admin user, **disabled by default** (az acr update --admin-enabled true). One account, full push and pull, password kept in App Service config. Dev and test only.
- **New tag** - az webapp config container set restarts the app and pulls it. Pushing again to the SAME tag is NOT detected: restart, or enable continuous deployment (ACR webhook on push).

KEDA scale rule

- **azure-servicebus** -> queueName, namespace, **messageCount**
- **azure-queue** (Storage) -> queueName, accountName, **queueLength**
- Polling interval **30 s**. Cool-down **300 s**, and only when scaling in from the final replica to 0.
- $\text{desiredReplicas} = \text{ceil}(\text{currentMetricValue} / \text{targetMetricValue})$
- minReplicas 0 = scale to zero. maxReplicas default **10**, configurable to 1,000.

App Service Key Vault refs

- **Syntax** - setting value `@Microsoft.KeyVault(...)` holding either `SecretUri=<full secret URI>` or `VaultName=myvault;SecretName=mysecret`. Version is optional.
- **Access** - managed identity on the app (system-assigned by default) with **Key Vault Secrets User** (RBAC) or Get on secrets (access policy).
- **No code change** - the app reads the resolved value as a normal environment variable. Works in app settings and connection strings.
- **Rotation** - no version in the URI -> latest version, refreshed within **24 hours**. Any config change restarts the app and refetches at once.
- **Pick one** - non-sensitive value (a model version) -> plain app setting. Password or API key -> Key Vault reference.

Kubernetes Service types

List them. Which is the default? Is Ingress one of them?

Answer on the reverse.

Rolling update defaults

maxSurge and maxUnavailable defaults - for a Deployment, a DaemonSet, and an AKS node-pool upgrade.

Answer on the reverse.

Kubernetes probes

What does each probe failure actually do? What makes a healthy pod unreachable?

Answer on the reverse.

Container Apps revisions

What are the revision modes? What is required before traffic weights work?

Answer on the reverse.

Rolling update defaults

- **Deployment:** maxSurge 25%, maxUnavailable 25%. Capacity never drops below 75%.
- **DaemonSet:** maxUnavailable **1**, maxSurge **0**.
- **AKS node pool upgrade:** max surge **1 node**, max unavailable **0**.
- Three different sets inside one exam's scope. Read which object is being upgraded before answering.

Kubernetes Service types

- **ClusterIP** - the default. Internal only.
- **NodePort** - opens a port on every node.
- **LoadBalancer** - provisions an Azure load balancer with a public IP.
- **ExternalName** - the fourth one people forget.
- **Ingress is NOT a Service type.** It is a separate object doing HTTP host / path / TLS routing, backed by an ingress controller.

Container Apps revisions

- **Single** (default) - one active revision, all traffic switches at once, weights unavailable.
- **Multiple** - revisions coexist, you set the weights.
- **Deployment labels (preview)** - a documented third mode.
- Traffic **weights require Multiple** mode and must total 100.
- A **revision label** gives one revision its own stable URL with no production traffic diverted.

Kubernetes probes

- **Liveness** fails -> kubelet terminates the container, restart policy applies.
- **Readiness** fails -> pod IP removed from Endpoints, but **not** restarted.
- **Startup** - holds the other two off while a slow app boots.
- Two ways a healthy app is unreachable: **selector mismatch** (Endpoints empty) or **pods not Ready**. Diagnose with `kubectl get endpoints`.

Container Apps scale setup

What goes into a custom scale rule? When is min replicas 1 better than 0? Which rules can never reach zero?

Answer on the reverse.

Container Apps secrets

Where are Container Apps secrets scoped? How do you point one at Key Vault, and how does the container read it?

Answer on the reverse.

AKS manifests and rollouts

What format is a Kubernetes manifest and which command applies it? Order the steps of a zero-downtime image update.

Answer on the reverse.

AKS autoscalers

HPA, VPA, cluster autoscaler, KEDA - what does each one scale? What must containers define before HPA works?

Answer on the reverse.

Container Apps secrets

- **Scope** - app level, shared by revisions. Adding or changing a secret does **NOT** create a revision: restart the revision or deploy a new one.
- **Key Vault reference** (preferred in production) - the secret holds keyVaultUrl = the Key Vault **secret** URI (.../secrets/name) + identity (system, or a user-assigned resource ID).
- **Identity** - enable managed identity and grant **Key Vault Secrets User**. At az containerapp create use a **user-assigned** identity; the system one does not exist yet.
- **Rotation** - no version in the URI -> latest version within **30 minutes**; revisions using it in env vars restart automatically.
- **Read it** - env var value secretref:<secret-name>, a scale rule auth, or a Secret volume (one file per secret).
- **Trap**: an API key is stored as a Key Vault **secret**, NOT a Key Vault key (keys are cryptographic keys).

Container Apps scale setup

- **Rule parts** - type = the KEDA scaler (azure-servicebus, azure-queue), metadata such as the **queue name**, and auth by secret or managed identity. Replica limits and polling are optional.
- **CLI** - --scale-rule-type, --scale-rule-metadata, --scale-rule-auth or --scale-rule-identity, plus --min-replicas / --max-replicas.
- **Min 0** - an idle app runs no replicas and has **no usage charges**; the first new message pays a cold start.
- **Min 1 or more** - one replica always running. Recommended for production to avoid **cold start** latency.
- **HTTP rule** - concurrent requests (default **10**); it does NOT watch a queue. With ingress and no rule -> default HTTP rule, 0 to 10 replicas.
- **CPU and memory** rules CANNOT scale to zero. Changing a scale rule creates a new revision.

AKS autoscalers

- **HPA** (HorizontalPodAutoscaler) - adds or removes **pod replicas** on CPU, memory or custom metrics, for example minReplicas 3, maxReplicas 10, averageUtilization 50.
- **HPA needs** CPU requests and limits on **all containers**. Check it with kubectl get hpa.
- **VPA** - right-sizes CPU/memory **requests/limits**; it does NOT add pods. Modes: Off, Initial, Recreate, InPlaceOrRecreate (AKS 1.34+); Auto is deprecated.
- **Cluster autoscaler** - adds **nodes** when pods are Pending for lack of resources.
- **KEDA** - event-driven, can scale to **zero**. Do NOT add a separate HPA to a workload KEDA already scales.
- **Manual** - kubectl scale --replicas=5.

AKS manifests and rollouts

- **Manifest** - a **YAML** file (NOT XML): apiVersion apps/v1, kind Deployment, metadata, spec with replicas, selector.matchLabels (must match the template labels) and containers.
- **Declarative** - kubectl apply -f deployment.yaml creates OR updates; Kubernetes reconciles the cluster to the desired state.
- **Rolling update** - change the image tag in the manifest -> kubectl apply -f -> kubectl rollout status. Pods are replaced gradually, no downtime.
- **Roll back** - kubectl rollout undo. The Recreate strategy replaces everything at once with a brief outage. Do NOT delete the Deployment or scale to 0 to update.
- **Secrets** - never hardcode them in a manifest; reference a Kubernetes Secret with secretKeyRef.

AKS pod troubleshooting

A pod keeps restarting or its Service returns nothing. What do you check first, and what does each status mean?

Answer on the reverse.

Cosmos consistency levels

All five, strongest to weakest. Which is default? What are the bounded-staleness minimums?

Answer on the reverse.

Cosmos vector indexes

The three types, their max dimensions, and the threshold before an index is used.

Answer on the reverse.

Cosmos RU and throttling

What costs 1 RU? Autoscale range? What happens when you exceed provisioned throughput?

Answer on the reverse.

Cosmos consistency levels

- **Strong** - always the latest committed write. Not available with multiple write regions.
- **Bounded staleness** - lags by at most K versions or T time, whichever hits first.
- **Session** - the **default**. Read-your-own-writes within one session token.
- **Consistent prefix** - never out of order, no time bound.
- **Eventual** - weakest, no bound.
- Minimums: multi-region **100,000 write ops or 300 s**; single-region **10 ops or 5 s**.
- Only Strong and Bounded staleness bound cross-region lag.

AKS pod troubleshooting

- **Start here** - kubectl describe pod (events, probes, env, mounts) and kubectl logs. Diagnose BEFORE changing probes, replicas or nodes.
- **CrashLoopBackOff** - the app crashes. kubectl logs --previous shows the last crashed instance.
- **ImagePullBackOff** - image cannot be pulled: wrong path or tag, or no registry access. Events say why.
- **Pending** - cannot be scheduled; events show Insufficient cpu / memory.
- **Service, no traffic** - kubectl describe svc: compare Selector with kubectl get pods --show-labels, and check targetPort matches the container port.
- **Also** - kubectl get events, kubectl logs -c <container>, kubectl port-forward, kubectl top (metrics server).

Cosmos RU and throttling

- A point read of a **1 KB item by id and partition key** = **1 RU**. A write of the same item is roughly 5 RU.
- Autoscale scales between **10% and 100%** of the configured maximum.
- Exceeding throughput -> **HTTP 429**, status RequestRateTooLarge, header **x-ms-retry-after-ms** (milliseconds). The SDK retries automatically.
- **Shared** throughput is provisioned on the database, shared by up to 25 containers. **Dedicated** is on the container.

Cosmos vector indexes

- **flat** - brute force, full precision, 100% recall. Max **505** dimensions.
- **quantizedFlat** - compressed then brute force. Max **4096**.
- **diskANN** - graph-based approximate. Max **4096**. Best at scale.
- quantizedFlat and diskANN need **at least 1,000 vectors**, otherwise Cosmos full-scans.
- distanceFunction: **cosine** (default), dotproduct, euclidean.
- Policy settings cannot be edited in place.

Cosmos change feed

The two modes, their limits, and what the processor requires.

Answer on the reverse.

Cosmos indexing policy

What are the defaults, and how do you cut write RU cost?

Answer on the reverse.

Cosmos SDK object model

Name the three SDK objects in the order you get them. Which one holds shared throughput? Which one runs item CRUD?

Answer on the reverse.

Cosmos vector search

How do you store an embedding, keep vector writes cheap, and return the 5 nearest items? Which index does NOT help?

Answer on the reverse.

Cosmos indexing policy

- **indexingMode:** consistent (default, index updates synchronously with each write) or none.
- **includedPaths** default is `/*` - every property of every item.
- **excludedPaths** default contains `/_etag/?`
- The more specific path wins when a path is in both lists.
- To cut write RUs: exclude the paths no query filters on. Adding included paths while leaving `/*` in place is a **no-op**.

Cosmos change feed

- **Latest version mode** - the default. Current state of each changed item. **Deletes are not captured.** Unlimited retention.
- **All versions and deletes mode** - every create, update and delete. Requires **continuous backups**. NoSQL API only.
- Processor needs two containers: the **monitored** container (the data) and the **lease** container (continuation tokens plus partition ownership - this is what lets instances divide work).
- A **partitioned** lease container needs `/id` as its partition key.

Cosmos vector search

- **Store** each embedding as an **array of numbers**, NOT a Base64 string. The vector policy sets path, dataType, dimensions, distanceFunction.
- **Vector index** - makes VectorDistance faster and cheaper in RU. A **composite index** does NOT serve vector distance.
- **Exclude the vector path** (`/embedding/`) from the regular index - indexing it wastes storage and raises write RU.
- **Lower precision** - float16 cuts vector storage **50%**; quantizedFlat and diskANN compress vectors.
- **Query:** `SELECT TOP 5 c.id, VectorDistance(c.embedding, @q) AS score FROM c ORDER BY VectorDistance(c.embedding, @q)`.
- Strong consistency, OFFSET, DISTINCT, GROUP BY: no help. The vector policy is container setup, NOT part of the query.

Cosmos SDK object model

- **CosmosClient** - created first, from the account endpoint and key (or a connection string). It connects lazily, on the first operation.
- **Database** (Python **DatabaseProxy**) - `client.get_database_client("name")`, `.NET GetDatabase()`. Holds **shared throughput**: `ReadThroughputAsync`, `ReplaceThroughputAsync`.
- **Container** (Python **ContainerProxy**) - `database.get_container_client("name")`, `.NET GetContainer()`. Runs item **CRUD** and queries: `create_item`, `read_item`, `upsert_item`, `delete_item`, `query_items`.
- **Python order:** create the client -> name the database -> get the container -> run the query.
- `DatabaseProxy` and `ContainerProxy` are **NOT** constructed directly - the client hands them out. An indexing policy is container configuration, not an SDK object.

Change feed scale-out

What are the four processor parts? How do several instances share the work, and what do Python apps use instead?

Answer on the reverse.

pgvector operators and indexes

The three distance operators, their operator classes, and the dimension limits.

Answer on the reverse.

PgBouncer on Azure PostgreSQL

Ports, how to enable it, the pool mode default, and the tier restriction.

Answer on the reverse.

PostgreSQL app connections

How should app code authenticate, reuse connections, survive traffic spikes, and pass user input? Which settings are NOT the fix?

Answer on the reverse.

pgvector operators and indexes

- $\leftrightarrow$ Euclidean / L2 $\rightarrow$ vector_l2_ops
- $\Leftrightarrow$ **cosine** $\rightarrow$ vector_cosine_ops
- $\langle \# \rangle$ negative inner product $\rightarrow$ vector_ip_ops
- The index operator class **must match** the operator in the query, or the index is ignored.
- ivfflat and hnsw index up to **2,000 dimensions**. diskann comes from the separate **pg_diskann** extension.
- Allowlist the extension in azure.extensions as **vector**, not pgvector.
- B-tree is for metadata columns; a vector index is for the embedding column. Never the reverse.

Change feed scale-out

- **Parts** - monitored container, **lease container**, compute instance, delegate (your handler code).
- **Lease container** - state storage: which partitions each instance owns, plus a **checkpoint**. ONE owner per lease at a time.
- **Scale out** - more instances with the **same lease container**, the **same processor name** and **different instance names**. Leases spread evenly.
- **Push model** (processor or Azure Functions trigger) reacts to inserts and updates - no scheduled full scans or SELECT polling.
- The processor is **.NET and Java** only. Python and Node.js use the **pull model**: query_items_change_feed plus a continuation token.
- Strong consistency and autoscale throughput do NOT distribute work.

PostgreSQL app connections

- **Managed identity** - the app gets a Microsoft Entra token and passes it as the **password**. No stored credentials. Map it to a role with pgaadauth_create_principal.
- **Pooling library** (psycopg_pool, Npgsql built-in pool) - reuses warm connections. A new connection costs **50-200 ms**: TCP, TLS, auth and a new server process.
- **Set a maximum pool size** - cap it at what the server allows divided by app instances (1,000 / 10 $\rightarrow$ **100** each). Too large or unlimited overwhelms the server.
- **Parameterized query** - %s placeholders, values passed as a **tuple**: cur.execute(sql, (doc_id,)). NEVER string formatting or concatenation; hand-escaping quotes is NOT the fix.
- Raising max_connections or shared_buffers, or adding read replicas, does NOT remove connection setup cost. Retry with backoff is for transient faults.

PgBouncer on Azure PostgreSQL

- **Port 6432** for PgBouncer. Direct PostgreSQL is **5432**. Same hostname.
- Enable with the server parameter **pgbouncer.enabled = true**. No restart needed.
- Azure default **pool_mode = transaction** (open-source PgBouncer defaults to session).
- default_pool_size **50**, max_client_conn **5000**, server_idle_timeout **600**.
- **Not available on the Burstable tier**. Supported on General Purpose and Memory Optimized.
- Raising max_connections is not pooling - more connections means more contention.

pgvector filtered retrieval

Which column types? In what order do you build it?
Write the filtered top-5 query. What cuts CPU when filters narrow the rows?

Answer on the reverse.

PostgreSQL vector resources

Match each symptom to a resource. Which tier has the most memory per vCore? In what order do you prove a scale-up worked?

Answer on the reverse.

Redis eviction policies

All eight. What do the volatile ones ignore? What is Azure's default?

Answer on the reverse.

Expiration vs eviction

Different triggers, different settings. How do you tell which a question is about?

Answer on the reverse.

PostgreSQL vector resources

- **CPU-bound latency** (distance math, CPU over 70%) -> scale **vCores**.
- **Index working set no longer fits** (cache hit ratio below 99%, more disk reads) -> **more memory**.
- **Memory per vCore**: Burstable 2 GB, General Purpose 4 GB, **Memory Optimized 8 GB**.
- **Growing embeddings -> storage autogrow**. Premium SSD only; grows online unless it crosses 4,096 GiB.
- **Prove it**: baseline P95 plus EXPLAIN ANALYZE -> check memory pressure and cache hit ratio -> move to more memory per vCore -> rerun the same benchmark.
- max_connections and read replicas do NOT cut single-query latency. Backup retention adds no capacity.

pgvector filtered retrieval

- **Columns** - embedding as **vector(n)**, NOT varchar, text or JSON. Dates as **timestampz**, NOT free-form text. Typed columns filter and index reliably.
- **Build order** - create table -> **load data** -> create indexes -> ANALYZE -> query. Loading before indexing is faster and gives a better index layout.
- **Query** - WHERE department = 'Finance' ORDER BY embedding <=> \$1 LIMIT 5. Use the exact match: ILIKE 'fin%' matches more than Finance.
- **B-tree on filter columns** - PostgreSQL filters first, then computes distance only for the matching rows. Less CPU.
- **Vector index** (hnsu, ivfflat, diskann) on the embedding - fewer distance calculations. Either index alone can lower P95. More dimensions ADD compute.

Expiration vs eviction

- **Expiration** - per key, **time**-triggered. EXPIRE, PEXPIRE, EXPIREAT, SETEX, TTL, PERSIST. Absolute: the countdown starts when set and reads do not extend it. Sliding = re-issue EXPIRE on read.
- **Eviction** - instance-wide, **memory**-triggered. maxmemory plus maxmemory-policy.
- A **duration** in the stem -> expiration. **Memory pressure**, or which keys **survive** when space runs short -> eviction.
- "Keep frequently read keys in memory" depends on memory: ample -> **EXPIRE on access** (sliding); running short -> **eviction policy**.

Redis eviction policies

- volatile-lru, volatile-lfu, volatile-random, volatile-ttl
- allkeys-lru, allkeys-lfu, allkeys-random
- noeviction - writes fail with an error
- **volatile-* only consider keys that have a TTL**. If no key has one, nothing is evicted and writes start failing.
- Azure default: **volatile-lru**. For a pure cache, allkeys-lru or allkeys-lfu.
- **LRU ranks by recency of last access** (approximate, sampled via maxmemory-samples). LFU ranks by count.

Redis ports by tier

Which port belongs to which service and tier?

Answer on the reverse.

RediSearch vector search

Field types, algorithms, distance metrics, and the create-time constraints.

Answer on the reverse.

Redis cache freshness

Which tool gives a fixed lifetime, which keeps data fresh after a source change, and which cuts repeat database reads? What does NOT affect freshness?

Answer on the reverse.

Keeping hot keys in Redis

Frequently read keys should stay and rarely read keys should go. When is the answer EXPIRE, and when an eviction policy? What does PERSIST do?

Answer on the reverse.

RediSearch vector search

- Schema field types: TEXT, TAG, NUMERIC, GEO, **VECTOR**. Only VECTOR supports KNN.
- Algorithms: **FLAT** (exact brute force, 100% recall) and **HNSW** (approximate, sub-linear - the answer for "at scale").
- DISTANCE_METRIC: **COSINE**, L2, IP.
- Modules are enabled **at creation time only** and cannot be added later. **Not** on Flash Optimized.
- Requires the **Enterprise clustering policy** and the **noeviction** eviction policy.
- FT.CREATE backfills existing keys unless SKIPINITIALSCAN.

Redis ports by tier

- **Azure Managed Redis: 10000** - used for **both** TLS and non-TLS. It is not a TLS-specific port.
- **Azure Cache for Redis, Basic / Standard / Premium: 6380** TLS, 6379 non-TLS.
- **Enterprise and Enterprise Flash: also 10000.**
- So port 10000 identifies the service tier, not an encryption setting.

Keeping hot keys in Redis

- **EXPIRE on each read** (sliding expiration) - every hit pushes the TTL out, so hot keys stay and cold keys expire. Needs NO memory pressure.
- **Eviction policies** (allkeys-lru, volatile-ttl and the rest) act **ONLY** under memory pressure. Expiration removes keys proactively instead.
- No memory pressure described and hot keys must stay -> **EXPIRE** on access. Memory running out -> eviction policy.
- **volatile-ttl** evicts the keys with the shortest remaining TTL. It never extends a TTL.
- **Absolute** (fixed-window) expiry ignores reads - use it when the lifetime must be exact. **PERSIST** removes the TTL, so the key never expires.
- TTL returns **-1** when a key has no expiry and **-2** when the key does not exist.

Redis cache freshness

- **Cache-aside** - read Redis; on a miss, query the database and write the result with a TTL. Repeat requests skip the database.
- **TTL** - SETEX key 600 value, or HSET then EXPIRE key 600. **600 s** = 10 minutes, **86400 s** = 24 hours. The lifetime counts from when it is set.
- **Invalidate on write** - update the source, then DEL the related keys; the next read reloads. Find keys by pattern with **SCAN**, NOT KEYS (KEYS blocks).
- Low-effort freshness: **tune TTL values** or **invalidate when the source changes**. Either works on its own.
- NOT freshness tools: **resetting the TTL on every read** (keeps stale keys alive longer), eviction policies and more memory (memory pressure), RDB persistence (durability), keyspace notifications (NOT available in Azure Managed Redis).

Redis vector storage basics

How do you write an embedding to a hash, and why can a search return nothing? What host, port and TLS setting does Azure Managed Redis use?

Answer on the reverse.

Event Grid retry schedule

What is the actual back-off schedule, and what counts as success?

Answer on the reverse.

Event Grid permanent failures

Which status codes are never retried? Retry-policy limits? Dead-letter facts?

Answer on the reverse.

Event Grid filtering

The three filter kinds, and which can see inside the payload?

Answer on the reverse.

Event Grid retry schedule

- 10 s, 30 s, 1 min, 5 min, 10 min, 30 min, 1 hr, 3 hr, 6 hr, then **every 12 hours up to 24 hours**, with randomization.
- It does **not** settle into a 5-minute loop.
- Event Grid waits 30 s for a response before queuing a retry.
- Only **200-204** count as a successful delivery.

Redis vector storage basics

- **Store** - numpy float32 array -> **tobytes()** -> HSET doc:1 with a mapping of the embedding bytes plus metadata. Vector and metadata share one hash.
- **HSET only stores.** Search needs an index: create_index (FT.CREATE) with a **VECTOR** field - FLAT or HNSW, TYPE, **DIM**, DISTANCE_METRIC - and a key prefix such as doc:.
- **Data present, zero matches** -> no vector index. Create a FLAT or HNSW index on the embedding field. A metadata-only index, TTL, persistence or more memory do NOT fix it.
- **Query** - *=>[KNN 5 @embedding \$query_vec AS score] with **dialect 2**. The query vector must have the index DIM.
- **Connect** - host name.region.redis.azure.net, port **10000**, ssl=True. r.expire(key, 600) = 10 minutes.

Event Grid filtering

- **Event type** - exact whole-string match on eventType.
- **Subject** - subjectBeginsWith and subjectEndsWith only. Prefix and suffix, nothing else.
- **Advanced** - the **only** filter that reads inside **data**. Takes operatorType, key (dot notation), value(s).
- Operators: NumberGreaterThan(OrEquals), NumberLessThan(OrEquals), NumberIn / NotIn, **NumberInRange / NotInRange**, BoolEquals, StringContains / BeginsWith / EndsWith / In and their Not forms, IsNullOrUndefined, IsNotNull.
- Arrays need enableAdvancedFilteringOnArrays.

Event Grid permanent failures

- **Azure resource endpoint:** 400, 403, 413.
- **Webhook:** 400, 401, 403, 413.
- **404 IS retried.** 414 does not appear in the failure table at all.
- Retry policy: max delivery attempts **1-30, default 30**; event TTL **1-1440 min, default 1440**. First limit hit wins.
- Dead-lettering is **off by default** - without it, undelivered events are dropped. ~5 min delay to the container; dropped if the destination is unreachable for **4 hours**.

Service Bus settlement

The four actions. Which increments the delivery count?

Answer on the reverse.

Service Bus numbers

Lock duration, max delivery count, duplicate detection window, and the FIFO rule.

Answer on the reverse.

Event Grid topics and setup

Setup order for publishing your own events? Custom vs system vs partner topic vs domain? What does the handshake prove?

Answer on the reverse.

Service Bus queues vs topics

Which entities fan one message out so every consumer gets a copy? Which Service Bus features are NOT delivery entities?

Answer on the reverse.

Service Bus numbers

- Peek-lock duration default **1 minute**, maximum 5 minutes, renewable.
- Max delivery count default **10**. Exceeding it auto-dead-letters with reason **MaxDeliveryCountExceeded**.
- Duplicate detection: enabled at **creation time only**. Window default **10 minutes**, min 20 s, max 7 days. Matches on MessageId.
- **FIFO**: a queue guarantees enqueue and retrieval order, **not processing order** under competing consumers. Guaranteed ordered processing needs **sessions** (SessionId), also creation-time only.

Service Bus settlement

- **Complete** - deleted from the queue.
- **Abandon** - lock released, available immediately, delivery count **INCREMENTS**.
- **Defer** - stays but set aside, retrievable **only by sequence number**, delivery count does **NOT** increment.
- **Dead-letter** - moved to the \$deadletterqueue sub-queue.
- Letting the lock **expire** behaves like abandon. Lock **lost** (MessageLockLostException) does **not** increment.
- "Without increasing the delivery count" appears for one reason only: defer.

Service Bus queues vs topics

- **Queue** - point-to-point: each message goes to **one** consumer; competing consumers share the load. FIFO delivery.
- **Topic + subscriptions** - publish/subscribe. Each subscription gets its **own copy** and acts like an independent virtual queue. Fan-out needs **both**.
- **Rules and filters** - SQL, correlation or Boolean (TrueFilter default) criteria for what a subscription receives. **NOT** entities.
- **Dead-letter queue** - automatic sub-queue of every queue and subscription. Holds messages that cannot be delivered or processed.
- **Auto-forwarding** - moves messages from a queue or subscription to one other queue or topic in the same namespace. Moves, does not copy.
- **Sessions** - a receiver locks all messages with one SessionId. Ordering, NOT fan-out.

Event Grid topics and setup

- **Order**: register the **Microsoft.EventGrid** provider -> create a **custom topic** -> create an **event subscription** on it, pointing at the handler.
- **Custom topic** - your app posts its own events to it. **System topic** - Azure service events (for example Storage). **Partner topic** - third-party SaaS events.
- **Event domain** - one endpoint for **thousands** of related topics (multitenant apps). Not part of custom topic setup.
- **Validation handshake** - proves you own a webhook endpoint (validationCode, or validationUrl within **10 minutes**). The Functions Event Grid trigger does it for you.
- Handshake and **output batching** (off by default) do **NOT** keep rejected events. Only a **dead-letter destination** does.

ServiceBusProcessor setup

The four setup steps in order? What happens if a handler is missing, or the message handler throws?

Answer on the reverse.

Functions triggers and retries

Queue retry behavior, timer format, blob trigger latency, Cosmos trigger requirements.

Answer on the reverse.

HttpTrigger attribute

The authorization levels, the default, and how the method list behaves.

Answer on the reverse.

Choosing a Functions trigger

Which trigger for an immediate reply, background work, a schedule, new blobs, changed documents? Why not do slow work in HTTP?

Answer on the reverse.

Functions triggers and retries

- **Queue Storage trigger: 5 attempts total including the first** (maxDequeueCount in host.json), then the message goes to **<queueName>-poison**.
- **Timer trigger:** six-field NCRONTAB, **seconds first** (five-field is also accepted).
- **Classic blob trigger polls** - up to **10 minutes** latency on Consumption if idle. The Event Grid blob trigger is push-based and recommended.
- **Cosmos DB trigger:** reads the change feed, requires a **lease container**, surfaces no deletes in latest-version mode.

ServiceBusProcessor setup

- **Order:** new ServiceBusClient -> CreateProcessor(queueName) -> assign ProcessMessageAsync and ProcessErrorAsync -> **StartProcessingAsync**.
- **Both handlers are mandatory** - StartProcessingAsync throws InvalidOperationException if one is missing. ProcessErrorAsync receives unhandled exceptions.
- **AutoCompleteMessages** - default **true**: completes the message after the handler finishes. ReceiveMode defaults to **PeekLock**.
- Handler **throws** without settling -> message **abandoned**, NOT completed, whatever AutoCompleteMessages says.
- **Dead-lettering** is a per-message call (DeadLetterMessageAsync), NOT a setup step.

Choosing a Functions trigger

- **HTTP** - request and response: the choice when the caller needs a **synchronous** reply. Serverless APIs and webhooks; add Microsoft Entra sign-in with App Service authentication.
- **Queue Storage** - runs per queue message: background work. **Timer** - fixed schedule. **Cosmos DB** - inserts and updates via the change feed.
- **Event Grid** - pushed events such as Microsoft.Storage.BlobCreated. Low latency, **no polling**.
- **HTTP limit:** no response within **230 seconds** -> the load balancer returns **502**.
- **Slow work:** the HTTP function puts the payload on a queue and returns at once; a **queue-triggered** function does the work and gets the queue retries.

HttpTrigger attribute

- **Anonymous** - no key. **Function** - a function key or host key, via ?code= or the x-functions-key header. **Admin** - the master key.
- Unspecified in C# defaults to **Function**. But in the **Node.js v4** model the default is **anonymous**.
- Authorization is **not enforced locally**, whatever the level says.
- The method list is **exhaustive**: "post" alone means GET is rejected. Omit the list and all HTTP methods are allowed.

Triggers and bindings

How many triggers per function? What brings blob contents in without SDK code? Which bindings does each service offer?

Answer on the reverse.

Function app configuration

Where do per-environment values live? What is local.settings.json for? How does a Key Vault reference work and rotate?

Answer on the reverse.

Secure function deployment

Order the steps for a secure function app release. Which option deploys both infrastructure and code from source control?

Answer on the reverse.

Key Vault data-plane roles

Narrowest to broadest. Which one reads secret values? Which is a trap?

Answer on the reverse.

Function app configuration

- **App settings** - exposed to code as **environment variables**, encrypted at rest. Separate values per app (environment).
- **local.settings.json** - local runs **ONLY**; Azure reads app settings, NOT this file. Excluded by the default .gitignore.
- **Key Vault reference** -
@Microsoft.KeyVault(SecretUri=...) as the setting value. Code reads it as normal, **no code changes**.
- Needs a **managed identity** with **Key Vault Secrets User** (or a Get secrets access policy).
- **Rotation**: versionless reference -> new version within **24 hours**, no redeploy. A versioned URI stays pinned.
- **App Configuration** - shared non-secret settings and feature flags, labels per environment, read with a managed identity.

Triggers and bindings

- **Trigger** - makes the function run. Exactly **one** per function; it can also pass data in (a special kind of input binding).
- **Input binding** - passes data into the function, such as a blob's contents, with **no SDK client code**.
- **Output binding** - writes data out, such as a new queue message. It does NOT read anything.
- Bindings are **optional**; a function can have several. They avoid hardcoding access to other services.
- **Support**: Timer = trigger **ONLY**. Queue Storage = trigger + output, **no input**. HTTP = trigger + output (the response). Blob and Cosmos DB = trigger, input and output.

Key Vault data-plane roles

- **Key Vault Reader** - metadata only, cannot read secret values.
- **Key Vault Secrets User** - reads secret **contents**. This is the read-a-secret role.
- **Key Vault Secrets Officer** - any secret action including set and delete.
- **Key Vault Administrator** - all data-plane ops on keys, secrets and certificates.
- Also: Key Vault Purge Operator, Key Vault Data Access Administrator.
- **Key Vault Contributor** is control-plane - manages the vault, grants **no** access to secret values.
- The **secret/key/cert data-plane** roles work only on vaults using the **Azure RBAC** permission model.

Secure function deployment

- **Order**: storage account -> function app -> managed identity + Key Vault access -> app settings with Key Vault references -> deploy code.
- **Storage first** - every function app needs a storage account (Blob, Queue and Table), set in AzureWebJobsStorage.
- **Identity before references** - a reference resolves only once the identity can read the secret.
- **GitHub Actions** - YAML workflow in /.github/workflows: azure/login (OpenID Connect recommended), azure/arm-deploy for the **Bicep** file, Azure/functions-action for the code.
- **Not declarative**: Azure CLI commands are imperative steps. Core Tools publish and local Git push **code only** to an app that already exists.

Key Vault versions and rotation

What makes rotation work automatically? What is the rotation mechanism?

Answer on the reverse.

DefaultAzureCredential chain

The documented order in the Python SDK.

Answer on the reverse.

App Configuration

What identifies a key-value? Refresh defaults? Roles? What belongs where?

Answer on the reverse.

Key Vault access for an app

Put the setup steps in order. Which scope? Which permission sounds right but does NOT read a secret?

Answer on the reverse.

DefaultAzureCredential chain

- 1 Environment variables. 2 Workload Identity. 3 **Managed Identity**. 4 Shared Token Cache (Windows only). 5 Visual Studio Code. 6 Azure CLI. 7 Azure PowerShell. 8 Azure Developer CLI. 9 Interactive browser (disabled by default). 10 Broker (WAM).
- Each link is skipped silently when its prerequisites are absent. Managed identity fires in Azure, a developer tool fires locally, **your code does not change**.
- **AZURE_TOKEN_CREDENTIALS** can restrict the chain to prod or dev, or to one named credential.
- ManagedIdentityCredential fails on a laptop. ClientSecretCredential needs a stored secret.

Key Vault versions and rotation

- **Versionless read** (get_secret("name")) resolves to the **current** version -> picks up rotation with no redeploy.
- **Versioned read** (version="123") is pinned to an immutable version forever -> needs a manual change after every rotation.
- Rotation flow: set an **expiry date** -> 30 days before, Key Vault publishes **Microsoft.KeyVault.SecretNearExpiry** to Event Grid -> an event subscription calls an **Azure Function** -> the Function writes a **new version** with a fresh expiry.
- **Keys** have native autorotation policies. **Secrets do not** - only your code can change the credential on the far end.

Key Vault access for an app

- **Order** - enable a **system-assigned managed identity** -> grant it **Key Vault Secrets User** -> read the secret at runtime with the SDK (SecretClient).
- **Managed identity** - no client ID or client secret to store. System-assigned is tied to the resource and deleted with it.
- **Scope** - assign at the **vault**, one vault per app per environment. Key Vault Administrator at subscription scope is far too broad.
- **RBAC vault** - authorize with role assignments. Switching to Azure RBAC **invalidates** all access policy permissions. RBAC is the default for new vaults from API version 2026-02-01.
- **recover** - restores a **soft-deleted** secret. It does NOT grant read.
- **Read at runtime** - a secret copied into app settings at deploy time is injected at startup and never sees a rotation.

App Configuration

- A key-value is identified by **key plus label**. Label is the per-environment mechanism.
- Refresh interval default **30 seconds** (SetRefreshInterval; SetCacheExpiration is deprecated). Key Vault secret refresh is a **separate** interval with a 1-minute minimum.
- **Sentinel key** - one key you bump last; the provider watches it and reloads everything at once.
- Roles: **App Configuration Data Reader / Data Owner**.
- Endpoints, timeouts and **feature flags** -> App Configuration. Passwords, keys, connection strings -> **Key Vault** (referenced by content type).

Where secrets belong

Which store holds a credential, and what does it add?
Why not App Configuration or Kubernetes secrets?

Answer on the reverse.

App Configuration Python SDK

Which call reads one setting? Which lists many? What gives one key a value per environment, and what does NOT?

Answer on the reverse.

Dynamic configuration refresh

How does an app pick up changed settings with no restart and no remote call per request? What is the Python pattern?

Answer on the reverse.

KQL operator shapes

What does each operator do to the shape of the data?
What is the performance order?

Answer on the reverse.

App Configuration Python SDK

- **Client** - AzureAppConfigurationClient(base_url, credential), for example credential=DefaultAzureCredential().
- **get_configuration_setting(key=, label=)** - reads **ONE** setting by key and label. This is the read call.
- **list_configuration_settings(key_filter=, label_filter=)** - enumerates many settings.
list_labels() - lists labels only, no setting value.
- **Labels** give one key a value per environment. NOT **key prefixes** (a hierarchy with / or :), NOT **content type** (a MIME type such as JSON), NOT **resource tags** (metadata on the Azure resource).

Where secrets belong

- **Test** - does the value grant access to a resource or contain a credential? -> **Key Vault**. Passwords, API keys, connection strings, storage access keys.
- **Key Vault adds** - encrypted storage with FIPS-validated modules, access logs, expiration, rotation workflows, soft delete and purge protection.
- **App Configuration** - for a secret it stores only a **Key Vault reference** (a URI), never the value. The two services do not talk to each other; the app reads Key Vault with its own identity.
- **Kubernetes secrets** - scoped to one cluster, not a central lifecycle store. On AKS, mount Key Vault objects with the **Secrets Store CSI Driver** (supports autorotation).

KQL operator shapes

- **where** (equivalent to **filter**) - keeps only the ROWS that match a predicate.
- **summarize** - aggregates, and **destroys the input rows**. Only the by columns and aggregates survive.
- **project** - selects, renames **and can compute**; drops everything unlisted.
- **extend** - appends a computed column. No by clause, cannot aggregate.
- **distinct** dedupes. **join** correlates on operation_Id / OperationId. **render** draws a chart.
- Order: table -> **time filter** -> other filters -> join -> summarize. Time filters skip whole storage extents.

Dynamic configuration refresh

- **Cache + interval** - the provider loads settings once, serves them from memory and checks again after the refresh interval. Reading the store on **every request** adds latency and risks **throttling** or overage charges.
- **Python** - load(endpoint=..., credential=..., refresh_on=[WatchKey("sentinel")], refresh_interval=10), then call config.refresh(). It is a **no-op** until the interval elapses.
- **.NET** - ConfigureRefresh with Register("SentinelKey", refreshAll: true), plus app.UseAzureAppConfiguration(). Keys are NOT watched unless registered.
- **Refresh fails** -> the app keeps using the cached configuration.
- **Environment variables** are NOT dynamic - injected at startup. App Service restarts the app on a change; Container Apps creates a new revision.
- **Access** - managed identity plus App Configuration Data Reader, so no connection string is stored.

The two telemetry schemas

App Insights vs Log Analytics table and column names. What unit is duration?

Answer on the reverse.

KQL time windows and grouping

Write "the last 30 minutes". How many rows does summarize return? Which join kind do you get by default?

Answer on the reverse.

Azure Monitor query tools

One-off investigation of app logs from several services: which tool? What do the activity log and workbooks give you instead?

Answer on the reverse.

OpenTelemetry pipeline

The four parts and what each one owns. What breaks if one is missing?

Answer on the reverse.

KQL time windows and grouping

- **ago(30m)** - current UTC time minus a timespan: where timestamp > ago(30m). Literals: 30m, 1h, 2d, 10s, 100ms. No weeks.
- **summarize ... by col** - one row per **distinct by value**. No by clause -> **one** row. Aggregates: count(), avg(), min(), max(), percentile(duration, 95).
- **bin(timestamp, 5m)** - rounds down into fixed buckets: summarize count() by bin(timestamp, 5m) | render **timechart** (first column must be datetime).
- **join** - default kind is **innerunique** (dedupes the left side). Write kind=inner for a standard inner join. Put the smaller table on the left.

The two telemetry schemas

- **Application Insights:** requests, dependencies, traces, exceptions. Columns timestamp (**lowercase** - KQL is case-sensitive), duration, success, operation_Id, operation_Name, cloud_RoleName.
- **Log Analytics workspace:** AppRequests, AppDependencies, AppTraces. Columns TimeGenerated, DurationMs, Success, OperationId, OperationName, AppRoleName.
- **duration is in MILLISECONDS** - "greater than two seconds" is duration > 2000.
- cloud_RoleName / AppRoleName = which component emitted the telemetry.

OpenTelemetry pipeline

- **TracerProvider** - makes tracers, owns config (sampler, resource, processors). No configured provider/export pipeline -> spans do **not** reach Azure Monitor.
- **Exporter** - the destination. AzureMonitorTraceExporter plus a connection string.
- **Span processor** - connects provider to exporter and decides when export happens. **BatchSpanProcessor** = **asynchronous** batches on a background thread; **SimpleSpanProcessor** = **synchronous** per span.
- **Spans** - created in code with tracer.start_as_current_span().
- An exporter with no processor feeding it exports **nothing**.

Azure Monitor query tools

- **KQL in Log Analytics** - write and run log queries interactively over the workspace. The direct route for an ad hoc investigation.
- **Activity log** - **control-plane** operations: create a VM, change a key vault access policy, deployment errors. NOT application telemetry.
- **Workbooks** - combine text, log queries, metrics and parameters into **rich interactive reports**. Built for reuse; more work than one query.
- **Start with a table name** - it scopes the query and improves performance. == is case-sensitive, =~ is case-insensitive.

Tracing setup traps

Which actions make distributed traces appear and correlate? Which look related but do NOT create spans?

Answer on the reverse.

OpenTelemetry samplers

The real spec names, the Python names, and the environment-variable values.

Answer on the reverse.

Picking a trace sampler

Which sampler records every span? Which keeps 10 percent? What is NOT a sampler, and how does Azure Monitor sample?

Answer on the reverse.

Trace context propagation

What carries the trace across a service boundary, and what has to be installed?

Answer on the reverse.

OpenTelemetry samplers

- **Spec:** AlwaysOn, AlwaysOff, TraceIdRatioBased, ParentBased. Env-var values are **not** class names.
- **Python:** ALWAYS_ON, ALWAYS_OFF, TraceIdRatioBased(rate), ParentBased(root=...), ParentBasedTraceIdRatio(rate).
- **OTEL_TRACES_SAMPLER standard values** - always_on/off, trace_id_ratio, parentbased_always_on/off, parentbased_trace_id_ratio.
- **Azure Monitor values** - use microsoft.fixed_percentage or microsoft.rate_limited; ARG = ratio or max traces/sec.
- ParentBased honors the upstream decision. TraceIdRatioBased hashes the trace ID, so every service agrees.
- configure_azure_monitor defaults to a **RateLimitedSampler at 5 traces/second** since 1.8.6.

Tracing setup traps

- **Needed** - configure a **trace exporter** and **instrument** the code with the OpenTelemetry SDK (either order), then **redploy** the services.
- **Distro shortcut** - configure_azure_monitor() sets up tracer, meter and logger providers exporting to Application Insights, reading the connection string from APPLICATIONINSIGHTS_CONNECTION_STRING in the environment. Do NOT also call instrument() on a library it already supports.
- **OpenTelemetry View** - customizes **metrics**, such as histogram buckets. Creates no spans.
- **TelemetryClient.TrackEvent()** - classic Application Insights API. Writes a **custom event** (customEvents table), not an OpenTelemetry span.
- **SpanKind** - SERVER / CONSUMER -> requests table. CLIENT / INTERNAL / PRODUCER -> dependencies table.

Trace context propagation

- **W3C traceparent** header: 00-<32-hex trace id>-<16-hex span id>-<flags>
- OpenTelemetry's current context is **process-local**. It does **not** cross a socket by itself.
- You need an **instrumentation library** - RequestsInstrumentor().instrument() for the requests library, or a framework instrumentation to extract on the receiving side - or a manual propagate.inject(headers).
- configure_azure_monitor() enables the supported instrumentations for you.
- Without it, the downstream service starts a brand-new, unrelated trace. Creating spans is not propagation.

Picking a trace sampler

- **Every span** (local testing) -> AlwaysOn. Its argument is fixed at 1.0, nothing to set.
- **About 10 percent** -> TraceIdRatioBased with ratio **0.1** (range 0.0 to 1.0).
- **NOT samplers** - the span processor (BatchSpanProcessor) and the exporter. The sampler goes on the **TracerProvider**: TracerProvider(sampler=...).
- **Application Insights sampler** - ApplicationInsightsSampler(0.1) or sampling_ratio=0.1 keeps about 10 percent; traces_per_second caps volume. It does **NOT** honor upstream decisions - it hashes the trace ID, so give every service the **same sampler and rate**.
- **Metrics are never sampled**. Ingestion sampling is not recommended.