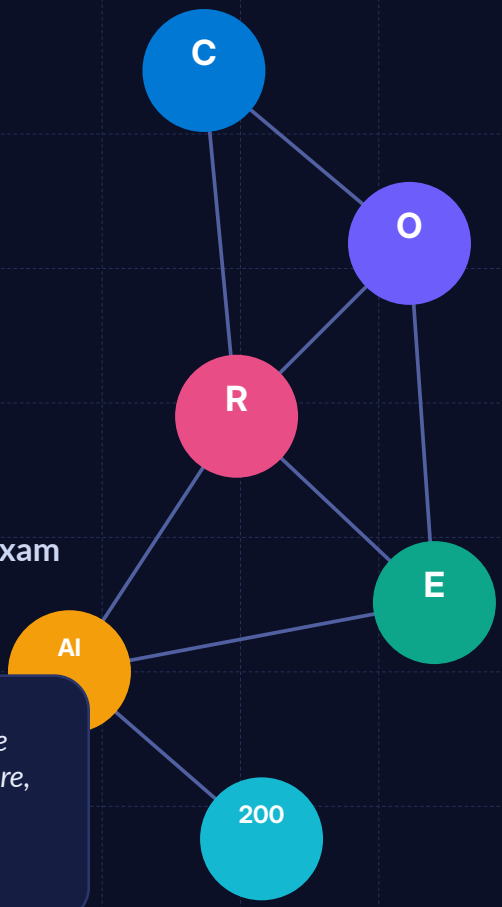
AI-200 • AZURE AI CLOUD DEVELOPER ASSOCIATE

THE CORE-4 FIELD MANUAL



A production-backend study system for the new AI-200 exam

AI-200 is not mainly about choosing a model. It tests whether the system around the model can package, retrieve, react, scale, secure, and explain itself.

120	700	4	PY	GA
MINUTES	PASS / 1000	DOMAINS	PYTHON	JULY 2026

CONTAINERS · VECTOR DATA · EVENT-DRIVEN · OBSERVABLE

Blueprint-aligned • Scenario-first • Hands-on • Exam traps decoded

TECH WITH KOUSHIK

Independent exam-preparation guide • Edition 1.0 • 14 July 2026

MISSION BRIEFING

What AI-200 actually tests — the CORE-4 operating model

Read every question as a broken production path. Find the layer that failed, then choose the smallest Azure mechanism that restores correctness.

The exam behind the exam

AI-200 is a back-end cloud engineering exam for AI workloads. The published blueprint emphasizes containers, vector-capable data services, messaging/eventing, Functions, secrets/configuration, distributed tracing, and KQL. Prompt engineering, model tuning, and agent orchestration are not standalone domains.

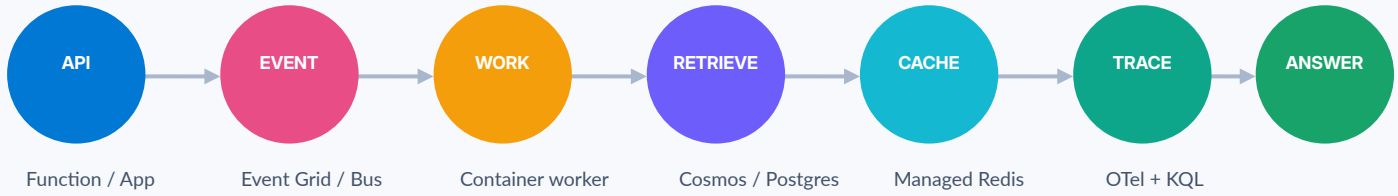
Your job: keep an AI request moving safely from ingress to data to compute — and make every failure diagnosable.

EXAM VITAL	FACT
Level	Intermediate
Role	Developer
Time	120 minutes
Pass	700 / 1000
Language	English (at publication)
Format	Proctored; interactive components may appear
Practice	Assessment not yet available on 14 Jul 2026

THE CORE-4 MAP

C	O	R	E
CONTAINERIZE	ORGANIZE DATA	REACT + CONNECT	ENFORCE + EXPLAIN
20–25%	25–30%	20–25%	20–25%

TRACE THE AI REQUEST — YOUR UNIVERSAL SCENARIO DECODER



Six meta-rules that decide the close questions

- Deterministic need → platform mechanism, not wishful code.
- Choose the service by message/data semantics, not by product popularity.
- Optimize the measured bottleneck: RU, index, connection, scaler, or dependency.
- Configuration is not a secret; a trace is not a log query.
- Prefer managed simplicity until the requirement explicitly needs control.
- For retries or replay, make handlers idempotent and preserve poison work.

THE BIG IDEA

Do not ask “Which Azure service knows AI?” Ask “Where is the request blocked — package, data, reaction, or observability?”

THE STUDY PLAN

The 4-week build plan — prove each domain in a working system

The fastest way to learn AI-200 is to assemble one small production spine and improve it each week.

WEEK	CORE PHASE	DO THIS	PROOF OF SKILL
1	C — Containers	Build a Python API image. Use ACR Tasks to build/push. Deploy once to App Service, then to Container Apps. Create a second revision and inspect logs/events.	You can explain image vs revision vs replica, and choose App Service, Container Apps, or AKS from requirements.
2	O — AI data	Store JSON + embeddings in Cosmos DB; measure RU. Run a pgvector query with a metadata filter. Add Redis caching and expiration.	A decision note comparing Cosmos DB, PostgreSQL, and Redis for the same retrieval workload.
3	R — Reactive spine	Queue work through Service Bus; fan out through a topic. Publish a custom Event Grid event with a filter. Build a Function trigger and binding.	A poison message reaches a DLQ; a filtered event reaches only the intended handler.
4	E — Secure + explain	Retrieve secrets from Key Vault, settings from App Configuration, emit OpenTelemetry, and answer a failure question with KQL. Take two timed mocks.	One end-to-end trace spans API → message → worker → database. You can find p95 latency and top failures in KQL.

BUILD PROOF

Use one repository with four folders: /api, /worker, /infra, /observability. Every study session should leave a durable artifact: Dockerfile, YAML, query, trace, or runbook.

READINESS GATE

Do not move on because a lab “worked.” Move on only when you can predict the failure mode and the first diagnostic surface without opening documentation.

ACCELERATED 10-DAY SPRINT

- **D1** Blueprint + CORE-4 map + service vocabulary
- **D2–3** ACR, App Service, Container Apps, KEDA, AKS
- **D4–6** Cosmos vector/RU/change feed + pgvector + Redis
- **D7** Service Bus + Event Grid + Functions
- **D8** Key Vault + App Configuration + OpenTelemetry + KQL
- **D9** Scenario drill #1; write why each distractor fails
- **D10** **Scenario drill #2 + 60-second brain dump + early rest**

SPACED REPETITION • Turn every Exam Signal and Trap Radar into a flashcard; review on days 2, 4, 8, 15, and 25.

CORE PHASE C • 20–25%

Containerize the AI workload

Build, store, deploy, scale, revise, and diagnose containers without reaching for Kubernetes by reflex.

1. Build and store

Azure Container Registry (ACR) is the private image store. ACR Tasks offloads build/test/push to Azure and can trigger on commits, schedules, or base-image updates.

Exam instinct: “No local Docker engine,” “rebuild after base patch,” or “cloud build pipeline” → ACR Tasks.

2. Host at the right abstraction

App Service: straightforward web/API container with managed web hosting.
Container Apps: microservices, revisions, event-driven scaling, scale-to-zero.
AKS: Kubernetes manifests, cluster-level control, custom operators/networking.

THE MANAGED-CONTROL LADDER

APP SERVICE
Web/API container; least operational overhead

→

CONTAINER APPS
Revisions + KEDA + microservices/event workers

→

AKS
Kubernetes API and cluster-level control

SCENARIO SIGNALS

- “Multiple versions, split traffic, rollback” → Container Apps revisions.
- “Scale from queue depth or event source” → KEDA rule matched to that source.
- “Deploy YAML manifests; inspect pods/services/events” → AKS.
- “App setting must appear as an environment variable” → platform app settings / secret reference.
- “Image builds after base image changes” → ACR Task dependency trigger.
- “Simple HTTP container; minimize operations” → App Service before AKS.

TRAP RADAR

A revision is an immutable version of a Container App. A replica is a running instance of that revision. Do not treat the words as interchangeable.

EXAM SIGNAL

A scaler is only correct when its metric represents pending work. Queue depth is a better scale signal for queue workers than CPU alone.

PRO MOVE

Use managed identity to pull from ACR and to retrieve Key Vault secrets. A secret reference is safer than baking credentials into an image or plain environment variable.

FIRST DIAGNOSTIC

Image pull failure → registry/identity. Crash loop → startup logs/config. No traffic → ingress/service/endpoints. No scaling → KEDA rule + authentication + source metrics.

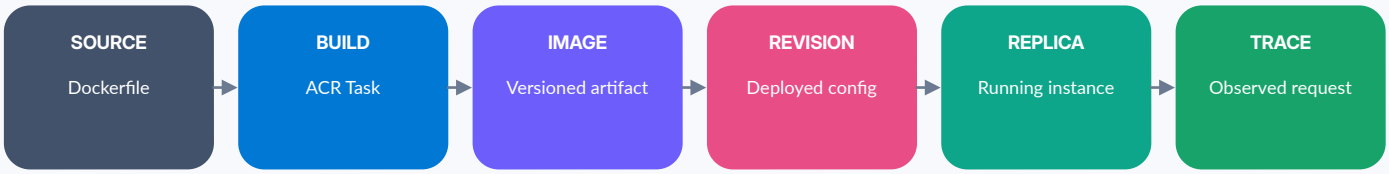
FAST RECOGNITION

The container decision lab — choose, release, scale, troubleshoot

Most container questions are a service-selection problem followed by a fault-isolation problem.

REQUIREMENT	BEST FIRST CHOICE	WHY / WATCH
Build and push in Azure; no local Docker	ACR Tasks	Cloud build; commit/base-image/schedule triggers.
Simple web/API container	App Service	Managed web hosting; settings become env vars; avoid AKS overhead.
Microservices, jobs, revisions, scale-to-zero	Container Apps	KEDA scaling + revision traffic/rollback.
Kubernetes manifests and full cluster control	AKS	Use when Kubernetes control is a requirement, not a preference.
Queue-driven worker	Container Apps + KEDA	Scale against Service Bus/Event Hubs/Redis or other event sources.
Safe progressive release	Container Apps multiple-revision mode	Route traffic between immutable revisions.
Inspect cluster failure	AKS logs + events + endpoints	Pod status alone does not prove service connectivity.

THE CONTAINER LIFECYCLE — KNOW THE NOUN AT EACH STAGE



KEDA: match the scaler to the backlog

HTTP scaler → concurrent requests.
Service Bus scaler → active message count.
CPU/memory scaler → resource pressure.
Custom scaler → external metric/event source.

Scale-to-zero saves cost, but cold starts and minimum replicas are design trade-offs.

The seven-step troubleshooting ladder

1. Image/tag exists and can be pulled.
2. Identity and registry permissions.
3. Environment variables and secret references.
4. Process starts and listens on expected port.
5. Ingress/service/endpoints route correctly.
6. KEDA source and auth are valid.
7. Trace the downstream dependency.

TRAP RADAR

More replicas do not fix an image, identity, port, DNS, or dependency failure. Scale only after the request path is valid.

BUILD PROOF

Deploy a broken image tag, a missing secret, and a wrong target port on purpose. Record the first log/event that reveals each failure.

CORE PHASE O • 25–30%

Build the AI data plane

This is the largest domain. Choose the store by data model, retrieval pattern, latency, and consistency — not because all three can hold vectors.

C

COSMOS DB

Distributed JSON truth

RU-based NoSQL + vector search + change feed

P

POSTGRESQL

Relational truth

SQL + transactions + pgvector + metadata filtering

M

MANAGED REDIS

Hot state

Cache + expiration + low-latency vector/full-text index

THE FOUR JOBS OF AN AI DATA PLANE

SOURCE OF TRUTH Durable operational state

VECTOR RETRIEVAL Nearest-neighbor search

HOT STATE Cache/session/recent results

CHANGE PROPAGATION React to inserts/updates

VECTOR RETRIEVAL ANATOMY

EXAM SIGNAL

“Global JSON, elastic throughput, RU, change feed” → Cosmos DB. “SQL joins/transactions + vector + metadata” → PostgreSQL. “Sub-millisecond hot cache/TTL” → Redis.

TRAP RADAR

A vector database choice is still a database choice. Ask where the authoritative record lives, how filters work, and what consistency/transaction guarantees are required.

PRO MOVE

Use metadata filters before or with vector ranking to reduce candidate work and prevent cross-tenant retrieval. The “best vector index” is workload-dependent: measure recall, latency, and cost together.

AI DATA PLANE

Cosmos DB — vector retrieval under an RU budget

The exam rewards candidates who understand that indexing, consistency, partitioning, and query shape are one cost-performance system.

RU economics — what changes the bill

- Point reads cost less than queries.
- More predicates, results, and cross-partition work increase query RU.
- Indexes accelerate reads but increase write cost and storage.
- Exclude unused paths to reduce write RU.
- Strong and bounded-staleness reads consume roughly twice the RUs of relaxed levels.
- Change feed and lease-container operations also consume RUs.

Vector policy + vector index

Vector policy defines path, datatype, dimensions, and distance function. Vector index makes VectorDistance queries lower-latency and lower-RU.

flat: exact brute-force, max 505 dimensions.
quantizedFlat: compressed brute-force, up to 4096 dimensions.
diskANN: approximate, high throughput/low RU at scale.

Policies/indexes are immutable after container creation.

CONSISTENCY — PICK THE WEAKEST LEVEL THAT SATISFIES THE BUSINESS RULE

STRONG	**Latest committed read globally**	Highest latency/RU trade-off
BOUNDED	**Staleness limited by time/versions**	Predictable lag
SESSION	**Read-your-writes in a session**	Common app default
PREFIX	**Ordered prefix; no gaps**	May lag
EVENTUAL	**Fastest/weakest freshness**	Converges later

Change feed processor — the moving cursor

Monitored container → lease container → workers → delegate. Leases coordinate ranges and checkpoints. Delivery is at least once, so handlers must be idempotent. A failed batch may replay; persist poison work instead of blocking the feed forever.

Python/Node commonly use the pull model for manual checkpoints; the processor library is available for .NET and Java.

VECTOR QUERY SHAPE

```
SELECT TOP 10 c.id, c.title
FROM c
WHERE c.tenantId = @tenant
ORDER BY VectorDistance(
  c.embedding, @queryVector
)
```

Filter + vector order. Keep vectors with source data to simplify hydration.

TRAP RADAR
"Add every field to the index" is not automatically an optimization. It can raise write RU and storage while adding no value to the actual query pattern.

AI DATA PLANE

PostgreSQL + pgvector and Azure Managed Redis

One is a relational system of record with vector operators; the other is a low-latency accelerator. Know the boundary.

PostgreSQL: vector search beside relational truth

Enable the extension as vector (the community project is called pgvector). Model metadata in normal columns/JSONB, store embeddings in vector(n), filter in SQL, and order by distance.

Use EXPLAIN (ANALYZE, BUFFERS) to verify whether an index is used. Tune compute, memory, storage, and connections together; built-in PgBouncer reduces connection churn.

Managed Redis: make the hot path faster

Use for cache-aside results, sessions, rate limits, short-lived features, and very low-latency search. Set expiration and invalidation deliberately. RediSearch adds secondary, full-text, and vector indexes.

Redis is usually not the authoritative durable record. A fast stale cache is still wrong.

PGVECTOR INDEX TRADE-OFFS

INDEX	STRENGTH	COST / TRAP
Exact scan	Perfect recall; no ANN index	Slow at scale; may parallelize.
IVFFlat	Fast build; lower memory	Needs training/data; tune lists/probes; lower speed-recall than HNSW.
HNSW	Strong speed-recall; can index empty table	Higher build time and memory; tune m / ef.
DiskANN	High recall/QPS with disk-efficient scale	Azure PostgreSQL-specific; still validate plan and recall.

SQL SHAPE

```
CREATE EXTENSION vector;
CREATE INDEX docs_hnsw
ON docs USING hnsw
(embedding vector_cosine_ops);

SELECT id, title
FROM docs
WHERE tenant_id = $1
ORDER BY embedding <=> $2
LIMIT 10;
```

Cache-aside in one sentence

Read cache → on miss read the source of truth → populate with TTL → invalidate or version on change.

Exam trap: setting a long TTL without an invalidation strategy for mutable AI context.

EXAM SIGNAL

“Too many short-lived database connections” → connection pooling/PgBouncer before simply raising max_connections.

PRO MOVE

Benchmark vector recall and latency after applying the exact metadata filter used in production; a global benchmark can hide the real workload.

CORE PHASE R • 20–25%

Build the reactive backbone

Service Bus carries durable work; Event Grid announces that something happened; Functions supply focused serverless execution.

Azure Service Bus — commands and work

Queue: point-to-point; one consumer completes each message.
Topic + subscriptions: one-to-many fan-out; each subscription receives its own copy and can filter.

Use peek-lock + Complete/Abandon/DeadLetter. Max delivery count defaults to 10 before DLQ. Inspect and reprocess poison messages deliberately.

Azure Event Grid — notifications and routing

Publish discrete events to interested handlers. Filter by event type, subject prefix/suffix, or advanced data fields. Push delivery is at least once and unordered; configure retries and Blob Storage dead-lettering when loss is unacceptable.

An event says “this happened,” not “perform this durable command exactly once.”

MESSAGE SEMANTICS DECIDER

DO THIS WORK	**Service Bus queue**
TELL MANY WORKERS	**Service Bus topic**
A RESOURCE CHANGED	**Event Grid**
RUN CODE NOW	**Function trigger**

AZURE FUNCTIONS — TRIGGER VS BINDING

Trigger

Exactly one trigger starts a function: HTTP, Service Bus, Event Grid, timer, Cosmos DB change feed, and more. The trigger is a special input binding.

Build serverless APIs with HTTP triggers; build reactive workers with event/message triggers.

Bindings

Input and output bindings declaratively connect to services and reduce plumbing code. They are optional; Azure SDK clients remain valid when you need fine-grained behavior.

Environment-specific host settings can be overridden through app settings.

TRAP RADAR

Retries imply duplicate delivery. Make consumers idempotent. “Exactly once” is usually achieved by business-level deduplication, not by assuming the broker never redelivers.

EXAM SIGNAL

“Blob created; route only .jpg files to a handler” → Event Grid subscription filter. “Process every order reliably” → Service Bus queue.

FAST RECOGNITION

The messaging decision lab — delivery, retry, poison work

The right answer preserves intent: durable command, broadcast work, or best-effort event notification.

SCENARIO	CHOOSE	WHY / FAILURE PATH
A single worker must process each invoice	Service Bus queue	Competing consumers; complete/abandon; DLQ after repeated failure.
Billing, analytics, and audit each need a copy	Service Bus topic + subscriptions	Durable fan-out; subscription filters.
React when a Blob is created	Event Grid	Native event routing; filter by type/subject/data.
Handler is temporarily unavailable	Retry + dead-letter	Event Grid can retry and dead-letter to Blob; Bus message remains durable.
Message repeatedly fails business validation	Dead-letter with reason	Do not hot-loop a non-retryable poison message.
HTTP endpoint becomes a serverless API	HTTP-triggered Function	Trigger starts code; output binding or SDK writes downstream.

THE RELIABLE CONSUMER LOOP

Poison message playbook

Transient dependency error → abandon/retry with backoff.
Validation/business violation → dead-letter with reason and correlation ID.
Expired work → TTL + dead-letter-on-expiration when review is required.
After repair → inspect, correct, and resubmit explicitly.

Never leave the DLQ unmonitored; it is a queue, not a trash can.

Event Grid delivery truth

Push delivery is at least once; ordering is not guaranteed. Filters reduce noise before the handler. Retry policy controls attempts/TTL. Dead-letter destination is Blob Storage. If no dead-letter is configured, undelivered events can be dropped after retry policy is exhausted.

PRO MOVE

Carry a correlation ID from the original API request into the message/event and the worker trace. Without it, the system may be “instrumented” but still impossible to debug end-to-end.

CORE PHASE E • 20–25%

Secure configuration, secrets, and the evidence trail

Key Vault protects sensitive values. App Configuration centralizes non-secret behavior. OpenTelemetry carries context. KQL turns telemetry into an answer.

Key Vault — sensitive objects

Store secrets, keys, and certificates. Prefer managed identity and RBAC over hard-coded credentials. Rotate secrets/keys and keep old versions only as long as required. App Service, Functions, and Container Apps can resolve Key Vault references.

Exam instinct: password/API key/certificate/private material → Key Vault.

App Configuration — application behavior

Centralize key-values, labels, feature flags, and immutable snapshots. Refresh settings without redeploying when the provider supports it. Key Vault references store a URI — the application still authenticates to both services.

Exam instinct: endpoint, feature flag, threshold, model name, rollout setting → App Configuration.

CONFIGURATION IS NOT A SECRET

VALUE	HOME	WHY
Database password	Key Vault	Sensitive; rotate and audit access.
Feature flag	App Configuration	Dynamic behavior, targeted rollout.
API endpoint	App Configuration	Non-secret environment setting.
Certificate/private key	Key Vault	Cryptographic material.
Release snapshot	App Configuration snapshot	Immutable, rollback-friendly config set.

OpenTelemetry — carry the story

Instrument traces, metrics, and logs with a common context. A trace follows one request through services; spans represent operations/dependencies. Propagate context across HTTP and messages. Export through Azure Monitor OpenTelemetry Distro / Application Insights.

Do not log sensitive prompts, secrets, or customer data by default.

KQL — interrogate the evidence

Use where for time/error filters, summarize for counts/percentiles, project for fields, join for cross-table correlation, and bin for time series. Scope matters: a resource-level Logs blade may hide records from other services.

KQL analyzes telemetry after it is collected; it does not instrument the app.

TRAP RADAR

Moving a password from code into App Configuration does not make it a secret.

EXAM SIGNAL

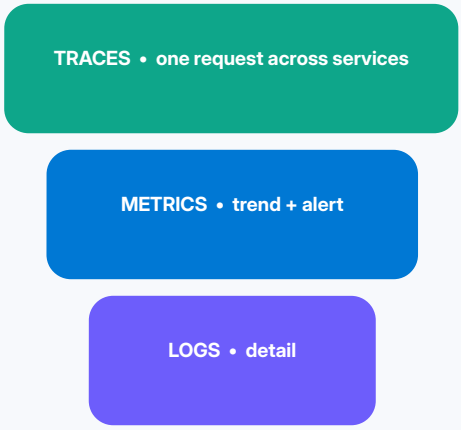
"Find where latency accumulated across services" → distributed trace, then KQL.

TRACE THE REQUEST

Diagnose end-to-end — from symptom to failing dependency

A log line tells you what one component said. A distributed trace tells you where the request spent time and where context broke.

THE TELEMETRY PYRAMID



The seven-hop trace check

1. Did ingress create/continue a trace?
2. Was trace context injected into Service Bus/Event Grid?
3. Did the worker extract it?
4. Which span has the largest duration?
5. Is the delay queue wait, compute, or dependency?
6. Do retries create duplicate child spans?
7. Can logs be joined by operation/correlation ID?

KQL PATTERNS TO RECOGNIZE

```
AppRequests
| where TimeGenerated > ago(1h)
| summarize requests=count(),
    failures=countif(Success == false),
    p95=percentile(DurationMs, 95)
  by OperationName, bin(TimeGenerated, 5m)
| order by p95 desc
```

```
where  → reduce rows
summarize → aggregate
percentile → tail latency
bin → time buckets
join → correlate tables
project → select/rename fields
```

Symptom → first evidence

High latency → trace waterfall + dependency duration.
Queue backlog → active-message metric + worker scale.
Throttling → Cosmos RU/429 metrics + query diagnostics.
Crash loop → container stdout/stderr + events.
Missing config → provider/identity logs + revision settings.

Instrumentation quality checks

Stable service.name / cloud role name.
Trace context propagated through messages.
No secrets or sensitive prompt bodies.
Sampling does not hide rare failures.
Business attributes are low-cardinality and safe.
Clock/time-zone handling is consistent.

PRO MOVE

Start with the request trace, then pivot to service metrics and component logs. Starting with millions of uncorrelated logs is slower and often points at the loudest component, not the failing one.

MYTH-BUSTING

The AI-200 myth vault + keyword decoder

These are the attractive wrong answers that sound technically advanced but violate the published scope or the scenario semantics.

"AI exam = prompt engineering."

The blueprint is mainly cloud back-end engineering around AI workloads.

"More indexing always lowers cost."

Indexes can raise write RU/storage; match actual queries.

"AKS is the most scalable, so choose it."

Choose AKS only when Kubernetes control/manifests are required.

"Scaling fixes timeouts."

Scaling cannot fix bad identity, DNS, ports, config, or dependency failures.

"Redis can replace the source of truth."

Redis accelerates the hot path; durability and invalidation still matter.

"App Configuration is secure storage for passwords."

Use Key Vault for secrets; App Configuration stores behavior/settings.

"Event Grid guarantees ordered delivery."

Push events are at least once and not ordered.

"KQL creates distributed traces."

OpenTelemetry instruments; KQL queries the collected evidence.

"Strong consistency is always safest."

It has latency/RU trade-offs; use only when the rule requires it.

"Retries make delivery exactly once."

Retries create duplicates; idempotency/deduplication preserves correctness.

KEYWORD → CONCEPT DECODER

base image patched → ACR Task trigger	revision traffic / rollback → Container Apps	manifest / pod / service → AKS
RU / partition / consistency → Cosmos DB	SQL + vector + metadata → PostgreSQL	TTL / hot cache → Managed Redis
one worker per message → Service Bus queue	fan-out durable copies → Service Bus topic	resource event / subject filter → Event Grid
starts a function → Trigger	declarative input/output → Binding	secret rotation → Key Vault
feature flag / snapshot → App Configuration	cross-service latency → OpenTelemetry trace	p95 / failures / time bucket → KQL summarize

EXAM SIGNAL

When two answers are technically possible, prefer the lower-operations service that directly satisfies the stated requirement — unless the question explicitly asks for deeper control.

ANSWER LOGIC

Twelve scenario drills — answer the requirement, not the product name

These are original blueprint-aligned drills, not recalled exam questions. Read the decisive clue in bold.

Q1 A CI runner has no Docker daemon, but must build and push an image.

ANSWER: ACR Tasks — cloud build/push.

Q2 A queue worker must scale from zero when Service Bus messages arrive.

ANSWER: Container Apps + KEDA Service Bus scaler.

Q3 A web API needs one managed container deployment with minimal ops.

ANSWER: App Service custom container.

Q4 A team must apply Kubernetes manifests and inspect pod events.

ANSWER: AKS.

Q5 Vector retrieval must join customer, invoice, and tenant metadata transactionally.

ANSWER: PostgreSQL + pgvector.

Q6 Global JSON items, RU budgeting, and reactions to item updates are required.

ANSWER: Cosmos DB + change feed.

Q7 Repeated answers need a 5-minute TTL and sub-millisecond access.

ANSWER: Managed Redis cache.

Q8 Each order must be processed once by one of many workers; poison orders must be preserved.

ANSWER: Service Bus queue + DLQ + idempotent consumer.

Q9 Three departments each need a durable copy of the same message.

ANSWER: Service Bus topic + three subscriptions.

Q10 Only image-created events under /uploads must reach the thumbnail handler.

ANSWER: Event Grid subscription filters.

Q11 A password must rotate without changing application code.

ANSWER: Key Vault + managed identity/reference.

Q12 Find which downstream call caused p95 latency across API, queue, worker, and DB.

ANSWER: OpenTelemetry distributed trace + KQL.

ANSWER METHOD

Underline the noun that defines semantics (command, event, revision, partition, secret, trace). Then reject every option that solves a different layer.

THE FINISH LINE

The 60-second brain dump + exam-day tactics

Write this map before question one. It prevents service-name drift when a scenario becomes long.

THE CORE-4 BRAIN DUMP

C ACR/Tasks → App Service / Container Apps / AKS → revision ≠ replica → KEDA signal → logs/events/connectivity

O Cosmos: RU + index + consistency + VectorDistance + change feed. Postgres: SQL + pgvector + pooling. Redis: TTL + hot path.

R Queue = one worker. Topic = durable fan-out. Event Grid = event/filter/retry. Function = one trigger + optional bindings.

E Secret → Key Vault. Setting/flag/snapshot → App Config. Trace → OTel. Analyze → KQL.

When time is tight

1. Read the final sentence first.
2. Identify the failing CORE layer.
3. Mark words like first, most secure, least operational overhead, or must guarantee.
4. Eliminate services that solve a neighboring layer.
5. For code/config, check trigger vs binding, secret vs setting, and index vs scan.
6. Flag and move when two options remain.

The "first fix" hierarchy

1. Correct wrong service semantics.
2. Correct identity/config/connection.
3. Correct index/query/scaler.
4. Add retry/DLQ/idempotency.
5. Add capacity only after evidence shows pressure.
6. Use deeper control (AKS/custom plumbing) only when the requirement demands it.

GO / NO-GO CHECKLIST

- [ ] I can choose among App Service, Container Apps, and AKS in <20 seconds.
- [ ] I can explain RU, indexing, consistency, and vector index trade-offs.
- [ ] I can choose Cosmos vs PostgreSQL vs Redis from one paragraph.
- [ ] I can distinguish a command queue, durable fan-out, and event notification.
- [ ] I can design a DLQ/idempotency path without hand-waving "exactly once."
- [ ] I can explain Key Vault vs App Configuration and OTel vs KQL.
- [ ] I can trace one request through API, message, worker, and database.
- [ ] I score ≥85% on two fresh scenario sets and can explain every miss.

YOU ARE READY WHEN YOU CAN TRACE, CHOOSE, AND EXPLAIN — NOT JUST RECOGNIZE LOGOS.

SOURCE TRAIL

Official blueprint and documentation used for this guide

All technical claims were checked against Microsoft Learn pages available on 14 July 2026. Features and exam objectives can change; verify the live blueprint before your exam.

1. **AI-200 study guide / skills measured**
   https://learn.microsoft.com/en-us/credentials/certifications/resources/study-guides/ai-200
2. **Azure AI Cloud Developer Associate certification**
   https://learn.microsoft.com/en-us/credentials/certifications/azure-ai-cloud-developer-associate/
3. **AI-200T00 course overview**
   https://learn.microsoft.com/en-us/training/courses/ai-200t00
4. **ACR Tasks overview**
   https://learn.microsoft.com/en-us/azure/container-registry/container-registry-tasks-overview
5. **Container Apps scaling / KEDA**
   https://learn.microsoft.com/en-us/azure/container-apps/scale-app
6. **Container Apps revisions**
   https://learn.microsoft.com/en-us/azure/container-apps/revisions-manage
7. **Monitor AKS**
   https://learn.microsoft.com/en-us/azure/aks/monitor-aks
8. **Cosmos DB RU and indexing**
   https://learn.microsoft.com/en-us/azure/cosmos-db/request-units
9. **Cosmos DB vector search**
   https://learn.microsoft.com/en-us/azure/cosmos-db/vector-search
10. **Cosmos DB change feed processor**
    https://learn.microsoft.com/en-us/azure/cosmos-db/change-feed-processor
11. **PostgreSQL pgvector**
    https://learn.microsoft.com/en-us/azure/postgresql/extensions/how-to-use-pgvector
12. **Optimize pgvector performance**
    https://learn.microsoft.com/en-us/azure/postgresql/extensions/how-to-optimize-performance-pgvector
13. **Azure Managed Redis modules**
    https://learn.microsoft.com/en-us/azure/redis/redis-modules
14. **Service Bus queues/topics/subscriptions**
    https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-queues-topics-subscriptions
15. **Service Bus dead-letter queues**
    https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dead-letter-queues
16. **Event Grid filtering**
    https://learn.microsoft.com/en-us/azure/event-grid/event-filtering
17. **Event Grid delivery and retry**
    https://learn.microsoft.com/en-us/azure/event-grid/delivery-and-retry
18. **Functions triggers and bindings**
    https://learn.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings
19. **Key Vault concepts and rotation**
    https://learn.microsoft.com/en-us/azure/key-vault/general/basic-concepts
20. **Azure App Configuration**
    https://learn.microsoft.com/en-us/azure/azure-app-configuration/
21. **OpenTelemetry in Application Insights**
    https://learn.microsoft.com/en-us/azure/azure-monitor/app/opentelemetry-enable
22. **Azure Monitor log queries / KQL**
    https://learn.microsoft.com/en-us/azure/azure-monitor/logs/get-started-queries

UPDATE NOTE

Blueprint version referenced: Microsoft Learn page updated 5 May 2026. At publication, the certification page listed the exam as generally available and the official Practice Assessment as not yet available. Recheck those two pages before publishing your video description.