Exam Code: AI-200
Exam Name: AI-200: Microsoft Certified: Azure AI Cloud Developer Associate (beta) Training Course
Certification: Microsoft Certified: Azure AI Cloud Developer Associate (beta)
Vendor: Microsoft

AI-200 Training Course

AI-200: Microsoft Certified: Azure AI Cloud Developer Associate (beta) Training Course

Structured Learning & Certification Preparation

Table of Contents

Introduction

This study pack is designed to support preparation for the Microsoft Certified: Azure AI Cloud Developer Associate (beta) exam through a clear, knowledge-point-driven structure. It brings the Azure AI cloud developer scope into one place so you can review containerized AI APIs, Azure data services for RAG and conversation state, Azure service integration, and secure observability-driven troubleshooting in the same order you are expected to master them.

The material is organized around 4 official blueprint domains, with each section keeping the detailed explanation content intact and pairing it with mapped practice questions. A practical way to use this pack is to move in a repeatable study, practice, and review cycle: study the explanation first, answer the related questions, then check the answer attachment to confirm where your understanding is already strong and where it still needs reinforcement.

About This Training / Certification

Microsoft Certified: Azure AI Cloud Developer Associate (beta) focuses on building, integrating, securing, monitoring, and troubleshooting Azure-hosted AI cloud workloads. The exam expects candidates to work with container images and Azure Container Apps, Azure AI Search, Cosmos DB, Blob Storage, Azure SDK clients, Azure AI service endpoints, Service Bus, Event Grid, managed identity, Key Vault, Application Insights, throttling behavior, and incident evidence.

This training content supports that preparation by keeping the knowledge explanations structured and by pairing each exam domain with directly mapped practice questions. The result is a study pack that helps you connect container runtime behavior, retrieval data design, service consumption patterns, identity boundaries, telemetry, and troubleshooting priority in a format that is practical for steady certification preparation.

What We Offer (AAAdemy)

AAAdemy provides structured training resources designed to support certification preparation and skill development across a wide range of IT domains. Our learning materials are built around clear knowledge structures, practical study guidance, and exam-oriented practice to help learners progress with confidence.

We offer well-organized knowledge explanations that break down complex topics into clear, understandable sections aligned with official exam objectives and real-world skill requirements. Each topic is designed to support both conceptual understanding and

practical application.

Our study plans and learning guidance help learners follow a logical progression, focusing on key concepts, common pitfalls, and effective preparation strategies. This approach enables learners to study efficiently while maintaining a clear view of their learning goals.

To reinforce understanding, AAAdemy also provides practice questions and exam-focused insights that reflect typical certification scenarios. These resources are intended to help learners evaluate their readiness and strengthen their confidence before taking an exam.

All content is designed for flexible, self-paced learning, allowing individuals to study independently or alongside their existing professional or academic commitments.

Knowledge Overview

- Develop containerized solutions on Azure
    - Dockerfile runtime selection, dependency installation, exposed port, and container startup validation
    - Registry login, repository tag selection, image pull identity, and image digest validation
    - Container app template, ingress target port, environment variables, and active revision routing
    - Replica limits, queue-driven scaling signals, concurrency, and cold-start impact on AI jobs
    - VNet integration, outbound reachability, DNS resolution, private endpoint access, and dependency timeout isolation
- Develop AI solutions using Azure data management services
    - Vector field dimensions, searchable text fields, semantic configuration, and vector search profile alignment
    - Data source connection, indexer status, skillset output mapping, and document key preservation
    - Hybrid query construction, vector field selection, filters, top-k retrieval, and answer grounding validation
    - Partition key design, point reads, request charge inspection, and conversation metadata lookup
    - Container access level, managed identity data access, blob metadata, and private document ingestion path
- Connect to and consume Azure services
    - Endpoint variables, client lifetime, retry policy, timeout settings, and API version selection
    - Endpoint URL, deployment name, API version, request body, throttling response, and model-call validation
    - Message contract, lock duration, retry handling, dead-letter reason, and idempotent worker execution
    - Event subscription filter, endpoint validation handshake, event schema, retry policy, and delivery metrics
    - HTTP client timeout, retry classification, secret retrieval, response contract, and failure isolation
- Secure, monitor, and troubleshoot Azure solutions
    - Identity assignment, token audience, RBAC scope, SDK credential chain, and runtime principal validation
    - Vault RBAC, secret URI, firewall path, private endpoint DNS, and secret client response validation
    - Request telemetry, dependency telemetry, exception capture, trace correlation, and operation id analysis
    - HTTP status classification, Retry-After handling, SDK retry policy, queue backpressure, and user-facing timeout control
    - Log source selection, metric dimension filtering, alert evidence, and root-cause timeline reconstruction

---

Detailed Knowledge Explanation

Develop containerized solutions on Azure

Dockerfile runtime selection, dependency installation, exposed port, and container startup validation

Exam Radar

Core Priority: This topic belongs to "Develop containerized solutions on Azure" and focuses on the working object behind the service name: container image.

High Frequency: Expect scenarios where the image builds locally but the cloud runtime exits before the AI endpoint becomes reachable. The best answer follows the failing runtime object, not the most visible Azure resource.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while base image, Python dependencies, startup command, exposed port, and registry tag is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when base image, Python dependencies, startup command, exposed port, and registry tag does not match the workload execution path.

Operational Dependency: The workload depends on base image, Python dependencies, startup command, exposed port, and registry tag. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: base image, Python dependencies, startup command, exposed port, and registry tag. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

A container image is the packaged runtime for the AI API: operating system layer, Python runtime, installed packages, application files, and startup command. Azure runs that image, not the project folder on the developer's machine.

Build success is only the first checkpoint. The image can build cleanly but still fail in Azure if the app listens on the wrong port, misses a native dependency, or starts with a command that exits after import.

For hands-on study, begin with container image: how it is named, how the app reaches it, and which field or status proves it is usable.

That order prevents cargo-cult troubleshooting. The command matters because it explains the symptom, not because it is a line to memorize.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |

| Dockerfile base image | Runtime layer | Python image, distro image, approved enterprise base | Undefined until `FROM` is declared | Native libraries and Python package compatibility | Image builds but app crashes on import or startup |

| Application listener | Bind address and port | 0.0.0.0 plus configured port | Framework default may differ | Container Apps target port and health probe | Ingress cannot route to the process |

| Hosting configuration | Ingress, traffic weight, replica bounds, or VNet setting | Enabled, disabled, internal, external, 0-100 traffic | Platform default unless explicit | Container app environment and revision template | Healthy resource still fails user traffic or dependency calls |

| Pull and network access | AcrPull, DNS, firewall, private endpoint, subnet route | Role assignment and route dependent | No proof until runtime access is tested | Managed identity and environment networking | Image pull failure, timeout, or endpoint resolution failure |

| Operational evidence | Revision state, log stream, queue metric, DNS test, or traffic split | Service-specific status output | Sparse until checked | Azure CLI, portal status, and diagnostics | Troubleshooting changes code without proving platform state |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is container image; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.
2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az acr repository show-tags --name ai200registry --repository ai-api --query "[0:5]"
   Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended container image and the service-specific attributes connected to base image, Python dependencies, startup command, exposed port, and registry tag.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   GET https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/rg-ai200/providers/Microsoft.ContainerRegistry/registries/ai200registry/repositories/ai-api/tags?api-version=2023-01-01-preview
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.
4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.
5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

The workload reaches Azure Container Registry and container runtime by reading configuration, choosing credentials, resolving the endpoint, and sending a request to container image. The service then checks authorization, object state, and request shape before it returns data or rejects the operation.

When base image, Python dependencies, startup command, exposed port, and registry tag is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Inspect container image | Run the Step 2 Azure CLI verification command | Output exposes the service state related to base image, Python dependencies, startup command, exposed port, and registry tag |

| Confirm service/API behavior | Run the Step 3 REST/API rehearsal request | Response code and body distinguish endpoint, authorization, object, and request-shape failures |
| Check authorization scope | `az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"` | Role scope is narrow enough and sufficient for the runtime path |

| Find application evidence | Application Insights > Transaction search > filter by operation id | Telemetry shows whether the dependency call happened and how it ended |
| Re-test original symptom | Repeat the original user action, queue message, event delivery, or API call | The same observable failure is gone after the targeted correction |

Push and Reference Images from Azure Container Registry

Registry login, repository tag selection, image pull identity, and image digest validation

Exam Radar

Core Priority: This topic belongs to "Develop containerized solutions on Azure" and focuses on the working object behind the service name: image repository tag.

High Frequency: A likely stem describes the deployment references an old or inaccessible image tag. The exam rewards evidence from image repository tag, not broad configuration changes.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while registry identity, repository permissions, tag naming, and pull authorization is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when registry identity, repository permissions, tag naming, and pull authorization does not match the workload execution path.

Operational Dependency: The workload depends on registry identity, repository permissions, tag naming, and pull authorization. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: registry identity, repository permissions, tag naming, and pull authorization. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

An ACR tag is a readable label such as `v1` or `prod`; a digest is the immutable content hash for the image manifest. Tags are convenient for humans, while digests prove the exact image content that was pulled.

This distinction is exam-relevant because a deployment can reference a familiar tag while the active revision still runs an older digest. When behavior does not match the expected code, verify tag and digest before changing application logic.

A practical learner should turn this topic into three questions: what selects image repository tag, what permission or route lets the app use it, and what evidence shows the call succeeded.

This sequence also keeps the learner from jumping to expensive fixes such as scaling, redeploying, or broadening permissions before the failed condition is known.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Image tag | Mutable label | v1, prod, build id, git SHA | Can move to a different digest | Push pipeline and deployment reference | Revision runs unexpected image content |

| Image digest | Immutable manifest hash | sha256 value | Generated on push | Registry manifest and layer set | Troubleshooting cannot prove which image actually ran |
| Hosting configuration | Ingress, traffic weight, replica bounds, or VNet setting | Enabled, disabled, internal, external, 0-100 traffic | Platform default unless explicit | Container app environment and revision template | Healthy resource still fails user traffic or dependency calls |

| Pull and network access | AcrPull, DNS, firewall, private endpoint, subnet route | Role assignment and route dependent | No proof until runtime access is tested | Managed identity and environment networking | Image pull failure, timeout, or endpoint resolution failure |
| Operational evidence | Revision state, log stream, queue metric, DNS test, or traffic split | Service-specific status output | Sparse until checked | Azure CLI, portal status, and diagnostics | Troubleshooting changes code without proving platform state |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is image repository tag; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.
2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az acr repository show --name ai200registry --image ai-api:v1 --query "{digest:digest,createdTime:createdTime}"
   Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended image repository tag and the service-specific attributes connected to registry identity, repository permissions, tag naming, and pull authorization.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   GET https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/rg-ai200/providers/Microsoft.ContainerRegistry/registries/ai200registry?api-version=2023-01-01-preview
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.
4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.
5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

At runtime, code does not consume a service name; it consumes a configured object. For Azure Container Registry, the request has to reach image repository tag, pass access checks, match the expected contract, and leave evidence in logs or status output.

When registry identity, repository permissions, tag naming, and pull authorization is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

Task	Precise Command or Path	Verification Standard
Inspect image repository tag	Run the Step 2 Azure CLI verification command	Output exposes the service state related to registry identity, repository permissions, tag naming, and pull authorization
Confirm service/API behavior	Run the Step 3 REST/API rehearsal request	Response code and body distinguish endpoint, authorization, object, and request-shape failures
Check authorization scope	`az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"`	Role scope is narrow enough and sufficient for the runtime path
Find application evidence	Application Insights > Transaction search > filter by operation id	Telemetry shows whether the dependency call happened and how it ended
Re-test original symptom	Repeat the original user action, queue message, event delivery, or API call	The same observable failure is gone after the targeted correction

Deploy an AI API to Azure Container Apps

Container app template, ingress target port, environment variables, and active revision routing

Exam Radar

Core Priority: This topic belongs to "Develop containerized solutions on Azure" and focuses on the working object behind the service name: container app revision.

High Frequency: When the stem says the container app exists but user traffic reaches an inactive or unhealthy revision, read it as an object-state problem first and a platform-change problem second.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while revision activation, ingress configuration, target port, environment variables, and health probe is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when revision activation, ingress configuration, target port, environment variables, and health probe does not match the workload execution path.

Operational Dependency: The workload depends on revision activation, ingress configuration, target port, environment variables, and health probe. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: revision activation, ingress configuration, target port, environment variables, and health probe. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

A Container Apps revision is an immutable snapshot of image, environment variables, ingress, probes, scale rules, and traffic assignment. Updating the template creates a new revision rather than quietly mutating the old runtime.

A container app can exist while the new revision is unhealthy or receives zero traffic. For troubleshooting, revision health and traffic weight are more useful than simply proving the app resource exists.

Study this as a runtime story rather than a service definition. The app points at container app revision, Azure evaluates revision activation, ingress configuration, target port, environment variables, and health probe, and the result shows up as a status, log, metric, or response.

Once the object and access path are clear, the rest of the evidence has a place to attach: logs explain the call, metrics show pressure, and responses classify the failure.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Revision | Template snapshot | Active, inactive, failed, traffic-weighted | Created from deployment template | Image, env vars, probes, ingress, scale | New deployment exists but receives no traffic |

| Traffic split | User routing | 0-100 percent per revision | Often remains on prior revision | Active revision and ingress | Users continue hitting older behavior |
| Hosting configuration | Ingress, traffic weight, replica bounds, or VNet setting | Enabled, disabled, internal, external, 0-100 traffic |

Platform default unless explicit | Container app environment and revision template | Healthy resource still fails user traffic or dependency calls |

| Pull and network access | AcrPull, DNS, firewall, private endpoint, subnet route | Role assignment and route dependent | No proof until runtime access is tested | Managed identity and environment networking | Image pull failure, timeout, or endpoint resolution failure |
| Operational evidence | Revision state, log stream, queue metric, DNS test, or traffic split | Service-specific status output | Sparse until checked | Azure CLI, portal status, and diagnostics | Troubleshooting changes code without proving platform state |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is container app revision; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.
2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az containerapp revision list --name ai-api --resource-group rg-ai200 --query "[].{name:name,active:properties.active,traffic:properties.trafficWeight,health:properties.healthState}"
   Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended container app revision and the service-specific attributes connected to revision activation, ingress configuration, target port, environment variables, and health probe.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   GET https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/rg-ai200/providers/Microsoft.App/containerApps/ai-api/revisions?api-version=2024-03-01
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.
4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.
5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

The execution chain is concrete: configuration selects container app revision, identity or key proves access, networking reaches the endpoint, and the service validates the request against its current state.

When revision activation, ingress configuration, target port, environment variables, and health probe is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |

| Inspect container app revision | Run the Step 2 Azure CLI verification command | Output exposes the service state related to revision activation, ingress configuration, target port, environment variables, and health probe |

| Confirm service/API behavior | Run the Step 3 REST/API rehearsal request | Response code and body distinguish endpoint, authorization, object, and request-shape failures |
| Check authorization scope | `az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"` | Role scope is narrow enough and sufficient for the runtime path |

| Find application evidence | Application Insights > Transaction search > filter by operation id | Telemetry shows whether the dependency call happened and how it ended |
| Re-test original symptom | Repeat the original user action, queue message, event delivery, or API call | The same observable failure is gone after the targeted correction |

Scale Containerized AI Workers on Azure

Replica limits, queue-driven scaling signals, concurrency, and cold-start impact on AI jobs

Exam Radar

Core Priority: This topic belongs to "Develop containerized solutions on Azure" and focuses on the working object behind the service name: scale rule.

High Frequency: This topic often appears as a small production incident: queued AI jobs age even though the worker application is deployed. The useful option is the one that proves the next dependency in the chain.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while min replicas, max replicas, scaler metadata, queue length, and worker concurrency is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when min replicas, max replicas, scaler metadata, queue length, and worker concurrency does not match the workload execution path.

Operational Dependency: The workload depends on min replicas, max replicas, scaler metadata, queue length, and worker concurrency. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: min replicas, max replicas, scaler metadata, queue length, and worker concurrency. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

A scale rule connects a workload signal to replica count. For AI workers, that signal may be queue depth, HTTP concurrency, CPU, or a custom scaler, but it must represent real pending work.

More replicas do not automatically fix AI throughput. Model throttling, queue lock duration, worker concurrency, and cold start decide whether scale-out helps or just creates more failed attempts.

The compact mental model is: selected object, access path, accepted request, observable result. For this topic, all four revolve around scale rule.

The exam skill is choosing the first useful observation. A fix that happens before that observation is usually only a guess.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Scaler metadata | Trigger configuration | Queue name, namespace, threshold, auth | No workload-aware scaling without rule | Metric source and permission | Backlog grows while replica count stays flat |

| Worker concurrency | Parallel jobs per replica | Single, batched, or configured concurrency | Defined by worker code | Lock duration and model latency | Duplicate work or lock expiration |
| Hosting configuration | Ingress, traffic weight, replica bounds, or VNet setting | Enabled, disabled, internal, external, 0-100 traffic | Platform default unless explicit | Container app environment and revision template | Healthy resource still fails user traffic or dependency calls |

| Pull and network access | AcrPull, DNS, firewall, private endpoint, subnet route | Role assignment and route dependent | No proof until runtime access is tested | Managed identity and environment networking | Image pull failure, timeout, or endpoint resolution failure |
| Operational evidence | Revision state, log stream, queue metric, DNS test, or traffic split | Service-specific status output | Sparse until checked | Azure CLI, portal status, and diagnostics | Troubleshooting changes code without proving platform state |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is scale rule; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.
2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az containerapp show --name ai-worker --resource-group rg-ai200 --query "properties.template.scale"
   Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended scale rule and the service-specific attributes connected to min replicas, max replicas, scaler metadata, queue length, and worker concurrency.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   GET https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/rg-ai200/providers/Microsoft.App/containerApps/ai-worker?api-version=2024-03-01
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.
4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.
5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

A user-visible result is the last link in the chain. Before that, Azure Container Apps scale rules has already evaluated the target object, the credential, the route, and the request contract.

When min replicas, max replicas, scaler metadata, queue length, and worker concurrency is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Inspect scale rule | Run the Step 2 Azure CLI verification command | Output exposes the service state related to min replicas, max replicas, scaler metadata, queue length, and worker concurrency |

| Confirm service/API behavior | Run the Step 3 REST/API rehearsal request | Response code and body distinguish endpoint, authorization, object, and request-shape failures |
| Check authorization scope | `az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"` | Role scope is narrow enough and sufficient for the runtime path |

| Find application evidence | Application Insights > Transaction search > filter by operation id | Telemetry shows whether the dependency call happened and how it ended |
| Re-test original symptom | Repeat the original user action, queue message, event delivery, or API call | The same observable failure is gone after the targeted correction |

Validate Container App Networking for AI Dependencies

VNet integration, outbound reachability, DNS resolution, private endpoint access, and dependency timeout isolation

Exam Radar

Core Priority: This topic belongs to "Develop containerized solutions on Azure" and focuses on the working object behind the service name: outbound dependency path.

High Frequency: Expect scenarios where the AI API times out only when calling private Azure services from the container runtime. The best answer follows the failing runtime object, not the most visible Azure resource.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while VNet configuration, private DNS zone, firewall rule, subnet route, and outbound endpoint is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when VNet configuration, private DNS zone, firewall rule, subnet route, and outbound endpoint does not match the workload execution path.

Operational Dependency: The workload depends on VNet configuration, private DNS zone, firewall rule, subnet route, and outbound endpoint. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: VNet configuration, private DNS zone, firewall rule, subnet route, and outbound endpoint. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

Container networking decides whether the running workload can resolve and reach private Azure services. Private endpoint success depends on DNS, routes, firewall rules, and VNet integration, not only on the target service being healthy.

Read the symptom carefully: timeouts often point to name resolution, routing, firewall, or private endpoint problems, while 401 and 403 usually point to identity or permission.

For hands-on study, begin with outbound dependency path: how it is named, how the app reaches it, and which field or status proves it is usable.

That order prevents cargo-cult troubleshooting. The command matters because it explains the symptom, not because it is a line to memorize.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Private DNS resolution | Name-to-address mapping | Private zone record or public endpoint | Depends on VNet links | Private endpoint and subnet DNS | Container times out or reaches public endpoint unintentionally |

| Outbound route | Runtime egress path | VNet route, firewall, NAT, service endpoint | Platform-dependent | Subnet and firewall policy | Service calls fail while local tests pass |
| Hosting configuration | Ingress, traffic weight, replica bounds, or VNet setting | Enabled, disabled, internal, external, 0-100 traffic | Platform default unless explicit | Container app environment and revision template | Healthy resource still fails user traffic or dependency calls |

| Pull and network access | AcrPull, DNS, firewall, private endpoint, subnet route | Role assignment and route dependent | No proof until runtime access is tested | Managed identity and environment networking | Image pull failure, timeout, or endpoint resolution failure |
| Operational evidence | Revision state, log stream, queue metric, DNS test, or traffic split | Service-specific status output | Sparse until checked | Azure CLI, portal status, and diagnostics | Troubleshooting changes code without proving platform state |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is outbound dependency path; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.
2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az containerapp exec --name ai-api --resource-group rg-ai200 --command "nslookup ai200-kv.vault.azure.net"
   Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended outbound dependency path and the service-specific attributes connected to VNet configuration, private DNS zone, firewall rule, subnet route, and outbound endpoint.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   GET https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/rg-ai200/providers/Microsoft.App/managedEnvironments/ai200-env?api-version=2024-03-01
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.

4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.

5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

The workload reaches Azure Container Apps networking by reading configuration, choosing credentials, resolving the endpoint, and sending a request to outbound dependency path. The service then checks authorization, object state, and request shape before it returns data or rejects the operation.

When VNet configuration, private DNS zone, firewall rule, subnet route, and outbound endpoint is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |

| Inspect outbound dependency path | Run the Step 2 Azure CLI verification command | Output exposes the service state related to VNet configuration, private DNS zone, firewall rule, subnet route, and outbound endpoint |

| Confirm service/API behavior | Run the Step 3 REST/API rehearsal request | Response code and body distinguish endpoint, authorization, object, and request-shape failures |

| Check authorization scope | `az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"` | Role scope is narrow enough and sufficient for the runtime path |

| Find application evidence | Application Insights > Transaction search > filter by operation id | Telemetry shows whether the dependency call happened and how it ended |

| Re-test original symptom | Repeat the original user action, queue message, event delivery, or API call | The same observable failure is gone after the targeted correction |

Practice Questions

1. A Python AI API container image builds successfully, but the Azure Container Apps revision exits immediately after startup. Which check should be performed first?
   A. Run the image locally and verify the startup command, listener port, and required environment variables.
   B. Increase the Container Apps maximum replica count.

C. Move the image to a different Azure Container Registry region.
D. Add a new Azure AI Search index for health checks.

2. A deployment references `ai-api:v2`, but the active app behavior matches an older source build. Which evidence most directly proves the exact image content running in the revision?
   A. The Container Apps environment name.
   B. The Azure Container Registry repository description.
   C. The image digest pulled by the active revision.
   D. The Dockerfile base image family.

3. A new Container Apps revision is created for an AI API, but users still receive responses from the previous version. What should the developer inspect first?
   A. The Azure OpenAI deployment temperature.
   B. The active revision traffic weights.
   C. The Cosmos DB partition key.
   D. The Service Bus lock duration.

4. An AI worker hosted in Container Apps processes queue messages slowly, and queue length keeps growing. Before changing replica limits, what should be validated?
   A. Whether the Blob Storage container allows public access.
   B. Whether the Azure AI Search semantic configuration is enabled.
   C. Whether the Event Grid endpoint validation event completed.
   D. Whether the scale rule can read the queue metric and maps to the intended workload signal.

5. A Container Apps-hosted AI API can call public endpoints but times out when reaching a private Key Vault endpoint. Which evidence should be collected first?
   A. DNS resolution, private endpoint routing, and firewall path from the app environment.
   B. A new Docker image tag with the same source code.
   C. A higher Azure OpenAI token limit.
   D. A different Service Bus retry policy.

6. A team wants to prove that a Container Apps revision is using the intended environment variables and ingress target port. Which object best represents the deployed template state to inspect?
   A. Azure Container Registry task.
   B. Container Apps revision.
   C. Azure AI Search indexer.
   D. Application Insights availability test.

7. A container app fails with an image pull authorization error against Azure Container Registry. Which remediation is most directly scoped to the failure?
   A. Enable semantic ranking on the search index.
   B. Increase the HTTP client timeout.
   C. Grant the hosting identity the AcrPull role on the registry scope.
   D. Rotate the Azure OpenAI deployment name.

8. An AI API container listens on port 8000, but Container Apps ingress is configured to target port 80. What is the likely result?
   A. Service Bus messages move directly to the dead-letter queue.
   B. The ACR digest changes automatically.
   C. Cosmos DB point reads become cross-partition queries.
   D. The app receives traffic but fails to route requests to the container listener.

9. A queue-driven AI worker has a minimum replica count of zero and users report delayed first processing after idle periods. Which factor should be considered first?
   A. Azure AI Search vector field dimension.
   B. Cold-start impact and scale-from-zero behavior.
   C. Key Vault secret version count.
   D. Application Insights sampling percentage only.
10. A developer wants to validate a container image before pushing it to ACR. Which action provides the most direct local evidence?
   A. Create an Event Grid subscription.
   B. Assign Key Vault Secrets User to the managed identity.
   C. Run the image locally with the expected command, port mapping, and environment variables.
   D. Increase Azure Container Apps CPU allocation.

Develop AI solutions using Azure data management services

Vector field dimensions, searchable text fields, semantic configuration, and vector search profile alignment

Exam Radar

Core Priority: This topic belongs to "Develop AI solutions using Azure data management services" and focuses on the working object behind the service name: vector index schema.

High Frequency: A likely stem describes retrieval fails or returns irrelevant content after embeddings are generated. The exam rewards evidence from vector index schema, not broad configuration changes.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while embedding dimension, vector profile, field name, document key, and semantic configuration is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when embedding dimension, vector profile, field name, document key, and semantic configuration does not match the workload execution path.

Operational Dependency: The workload depends on embedding dimension, vector profile, field name, document key, and semantic configuration. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: embedding dimension, vector profile, field name, document key, and semantic configuration. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

An Azure AI Search index schema defines which fields can be searched, filtered, returned, and compared as vectors. RAG quality depends on this schema because the model can only use the chunks returned by retrieval.

Vector field dimensions must match the embedding model output. If the schema expects one dimension count and the embedding has another, indexing or querying fails before the language model can produce a grounded answer.

A practical learner should turn this topic into three questions: what selects vector index schema, what permission or route lets the app use it, and what evidence shows the call succeeded.

This sequence also keeps the learner from jumping to expensive fixes such as scaling, redeploying, or broadening permissions before the failed condition is known.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Vector field | Dimension and type | Collection of numeric vector values | Absent unless declared | Embedding model output dimension | Indexing or query fails on dimension mismatch |

| Vector profile | Search algorithm binding | Named profile and algorithm config | No vector behavior without binding | Vector field schema | Vector query cannot use intended field |
| Ingestion path | Blob source, indexer run, skillset output, or write operation | Scheduled, manual, or SDK-driven | No freshness guarantee | Credentials, mappings, and source availability | New documents exist but are not searchable |

| Query or read evidence | Search result ids, RU charge, point read, selected fields, filters | Response-specific | Unknown until observed | Request body and API version | Model receives the wrong context or state lookup slows down |
| Security boundary | Data-plane role, storage access, private network, or admin key | Least-privilege role or key | Unsafe if made public | Managed identity and network path | Private data exposure or 403 during ingestion |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is vector index schema; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.
2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az search index show --service-name ai200-search --resource-group rg-ai200 --name docs-index --query "{name:name,fields:fields[].name,vectorProfiles:vectorSearch.profiles[].name}"
   Command note: Azure AI Search CLI support depends on installed extensions and current command surface. In a live lab, confirm with `az search --help`; when uncertain, use REST or portal status as the authoritative validation path.

Expected checkpoint: the output shows the intended vector index schema and the service-specific attributes connected to embedding dimension, vector profile, field name, document key, and semantic configuration.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   GET https://ai200-search.search.windows.net/indexes/docs-index?api-version=2024-07-01
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.

4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.
5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

At runtime, code does not consume a service name; it consumes a configured object. For Azure AI Search, the request has to reach vector index schema, pass access checks, match the expected contract, and leave evidence in logs or status output.

When embedding dimension, vector profile, field name, document key, and semantic configuration is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Inspect vector index schema | Run the Step 2 Azure CLI verification command | Output exposes the service state related to embedding dimension, vector profile, field name, document key, and semantic configuration |

| Confirm service/API behavior | Run the Step 3 REST/API rehearsal request | Response code and body distinguish endpoint, authorization, object, and request-shape failures |
| Check authorization scope | `az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"` | Role scope is narrow enough and sufficient for the runtime path |

| Find application evidence | Application Insights > Transaction search > filter by operation id | Telemetry shows whether the dependency call happened and how it ended |
| Re-test original symptom | Repeat the original user action, queue message, event delivery, or API call | The same observable failure is gone after the targeted correction |

Ingest Documents into Azure AI Search

Data source connection, indexer status, skillset output mapping, and document key preservation

Exam Radar

Core Priority: This topic belongs to "Develop AI solutions using Azure data management services" and focuses on the working object behind the service name: indexer execution.

High Frequency: When the stem says new source documents are not available to retrieval even though storage upload succeeded, read it as an object-state problem first and a platform-change problem second.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while data source credentials, indexer schedule, field mapping, skillset output, and indexing errors is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when data source credentials, indexer schedule, field mapping, skillset output, and indexing errors does not match the workload execution path.

Operational Dependency: The workload depends on data source credentials, indexer schedule, field mapping, skillset output, and indexing errors. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: data source credentials, indexer schedule, field mapping, skillset output, and indexing errors. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

An indexer is the ingestion worker that reads a data source, applies mappings or skillset output, and writes documents into an index. Uploading a blob only proves the source file exists; it does not prove the index contains searchable content.

Indexer status is the first stop when new documents are missing from search results. It exposes item failures, field mapping problems, authentication errors, and skillset output issues.

Study this as a runtime story rather than a service definition. The app points at indexer execution, Azure evaluates data source credentials, indexer schedule, field mapping, skillset output, and indexing errors, and the result shows up as a status, log, metric, or response.

Once the object and access path are clear, the rest of the evidence has a place to attach: logs explain the call, metrics show pressure, and responses classify the failure.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Indexer status | Execution state | Running, success, transient failure, permanent failure | No freshness proof until checked | Data source and skillset mapping | Uploaded files never become searchable |

| Field mapping | Source-to-index projection | Blob metadata, content, skill output | Implicit mapping may miss fields | Index schema and skillset output | Documents index without required searchable fields |
| Ingestion path | Blob source, indexer run, skillset output, or write operation | Scheduled, manual, or SDK-driven | No freshness guarantee | Credentials, mappings, and source availability | New documents exist but are not searchable |

| Query or read evidence | Search result ids, RU charge, point read, selected fields, filters | Response-specific | Unknown until observed | Request body and API version | Model receives the wrong context or state lookup slows down |
| Security boundary | Data-plane role, storage access, private network, or admin key | Least-privilege role or key | Unsafe if made public | Managed identity and network path | Private data exposure or 403 during ingestion |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is indexer execution; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.

2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az search indexer status --service-name ai200-search --resource-group rg-ai200 --name docs-indexer
   Command note: Azure AI Search CLI support depends on installed extensions and current command surface. In a live lab, confirm with `az search --help`; when uncertain, use REST or portal status as the authoritative validation path.

Expected checkpoint: the output shows the intended indexer execution and the service-specific attributes connected to data source credentials, indexer schedule, field mapping, skillset output, and indexing errors.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   GET https://ai200-search.search.windows.net/indexers/docs-indexer/status?api-version=2024-07-01
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.
4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.
5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

The execution chain is concrete: configuration selects indexer execution, identity or key proves access, networking reaches the endpoint, and the service validates the request against its current state.

When data source credentials, indexer schedule, field mapping, skillset output, and indexing errors is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

Task	Precise Command or Path	Verification Standard
Inspect indexer execution	Run the Step 2 Azure CLI verification command	Output exposes the service state related to data source credentials, indexer schedule, field mapping, skillset output, and indexing errors
Confirm service/API behavior	Run the Step 3 REST/API rehearsal request	Response code and body distinguish endpoint, authorization, object, and request-shape failures
Check authorization scope	`az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"`	Role scope is narrow enough and sufficient for the runtime path
Find application evidence	Application Insights > Transaction search > filter by operation id	Telemetry shows whether the dependency call happened and how it ended
Re-test original symptom	Repeat the original user action, queue message, event delivery, or API call	The same observable failure is gone after the targeted correction

Query Azure AI Search for Retrieval-Augmented Generation

Hybrid query construction, vector field selection, filters, top-k retrieval, and answer grounding validation

Exam Radar

Core Priority: This topic belongs to "Develop AI solutions using Azure data management services" and focuses on the working object behind the service name: retrieval query.

High Frequency: This topic often appears as a small production incident: the generated answer is fluent but not grounded in the expected document set. The useful option is the one that proves the next dependency in the chain.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while query text, vector payload, filter clause, selected fields, scoring profile, and top-k result set is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when query text, vector payload, filter clause, selected fields, scoring profile, and top-k result set does not match the workload execution path.

Operational Dependency: The workload depends on query text, vector payload, filter clause, selected fields, scoring profile, and top-k result set. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: query text, vector payload, filter clause, selected fields, scoring profile, and top-k result set. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

A RAG query is not just a prompt. It combines query text, vector payload, filters, selected fields, and top-k result count to decide which evidence reaches the generation step.

When an answer is fluent but ungrounded, inspect retrieval output before prompt wording. Wrong document ids, missing fields, or an over-restrictive filter leave the model with the wrong evidence.

The compact mental model is: selected object, access path, accepted request, observable result. For this topic, all four revolve around retrieval query.

The exam skill is choosing the first useful observation. A fix that happens before that observation is usually only a guess.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |

| Vector query | Retrieval payload | Vector, k, fields, filter | No grounding without returned documents | Index schema and embedding dimensions | Model receives irrelevant or empty context |

| Selected fields | Returned evidence | content, title, chunk id, source uri | May omit needed fields | Prompt assembly logic | Answer cannot cite or use the right source text |
| Ingestion path | Blob source, indexer run, skillset output, or write operation | Scheduled, manual, or SDK-driven | No freshness guarantee | Credentials, mappings, and source availability | New documents exist but are not searchable |

| Query or read evidence | Search result ids, RU charge, point read, selected fields, filters | Response-specific | Unknown until observed | Request body and API version | Model receives the wrong context or state lookup slows down |
| Security boundary | Data-plane role, storage access, private network, or admin key | Least-privilege role or key | Unsafe if made public | Managed identity and network path | Private data exposure or 403 during ingestion |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is retrieval query; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.

2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az rest --method post --uri "https://ai200-search.search.windows.net/indexes/docs-index/docs/search?api-version=2024-07-01" --headers "Content-Type=application/json" --body @query.json
   Command note: This is an Azure CLI REST wrapper used for rehearsal. Validate endpoint URL, admin/query key or token, and API version against current Azure AI Search documentation.

Expected checkpoint: the output shows the intended retrieval query and the service-specific attributes connected to query text, vector payload, filter clause, selected fields, scoring profile, and top-k result set.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   POST https://ai200-search.search.windows.net/indexes/docs-index/docs/search?api-version=2024-07-01
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.

4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.

5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

A user-visible result is the last link in the chain. Before that, Azure AI Search query API has already evaluated the target object, the credential, the route, and the request contract.

When query text, vector payload, filter clause, selected fields, scoring profile, and top-k result set is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |

| Inspect retrieval query | Run the Step 2 Azure CLI verification command | Output exposes the service state related to query text, vector payload, filter clause, selected fields, scoring profile, and top-k result set |

| Confirm service/API behavior | Run the Step 3 REST/API rehearsal request | Response code and body distinguish endpoint, authorization, object, and request-shape failures |
| Check authorization scope | `az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"` | Role scope is narrow enough and sufficient for the runtime path |

| Find application evidence | Application Insights > Transaction search > filter by operation id | Telemetry shows whether the dependency call happened and how it ended |
| Re-test original symptom | Repeat the original user action, queue message, event delivery, or API call | The same observable failure is gone after the targeted correction |

Persist AI Conversation State in Azure Cosmos DB

Partition key design, point reads, request charge inspection, and conversation metadata lookup

Exam Radar

Core Priority: This topic belongs to "Develop AI solutions using Azure data management services" and focuses on the working object behind the service name: conversation-state container.

High Frequency: Expect scenarios where conversation history reads become slow or expensive as tenants and sessions increase. The best answer follows the failing runtime object, not the most visible Azure resource.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while partition key, item id, tenant scope, request charge, and query filter is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when partition key, item id, tenant scope, request charge, and query filter does not match the workload execution path.

Operational Dependency: The workload depends on partition key, item id, tenant scope, request charge, and query filter. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: partition key, item id, tenant scope, request charge, and query filter. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

Conversation state usually follows tenant, user, conversation, session, or turn boundaries. Cosmos DB performance depends on choosing a partition key that matches those common reads and writes.

Request units are the operational evidence. A design that forces cross-partition scans may work in a demo but become slow or expensive as users and conversation history grow.

For hands-on study, begin with conversation-state container: how it is named, how the app reaches it, and which field or status proves it is usable.

That order prevents cargo-cult troubleshooting. The command matters because it explains the symptom, not because it is a line to memorize.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Partition key | Logical distribution | tenantId, userId, conversationId, composite model | Fixed after container creation | Read/write access pattern | Hot partition or cross-partition query cost |

| Request charge | RU evidence | Numeric charge per operation | Observed per request | Query shape and item size | Latency and cost grow without visible code error |
| Ingestion path | Blob source, indexer run, skillset output, or write operation | Scheduled, manual, or SDK-driven | No freshness guarantee | Credentials, mappings, and source availability | New documents exist but are not searchable |

| Query or read evidence | Search result ids, RU charge, point read, selected fields, filters | Response-specific | Unknown until observed | Request body and API version | Model receives the wrong context or state lookup slows down |
| Security boundary | Data-plane role, storage access, private network, or admin key | Least-privilege role or key | Unsafe if made public | Managed identity and network path | Private data exposure or 403 during ingestion |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is conversation-state container; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.

2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az cosmosdb sql container show --account-name ai200-cosmos --database-name appdb --name conversations --resource-group rg-ai200 --query "resource.partitionKey"
   Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended conversation-state container and the service-specific attributes connected to partition key, item id, tenant scope, request charge, and query filter.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   GET https://ai200-cosmos.documents.azure.com/dbs/appdb/colls/conversations
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.

4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.

5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

The workload reaches Azure Cosmos DB for NoSQL by reading configuration, choosing credentials, resolving the endpoint, and sending a request to conversation-state container. The service then checks authorization, object state, and request shape before it returns data or rejects the operation.

When partition key, item id, tenant scope, request charge, and query filter is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Inspect conversation-state container | Run the Step 2 Azure CLI verification command | Output exposes the service state related to partition key, item id, tenant scope, request charge, and query filter |

| Confirm service/API behavior | Run the Step 3 REST/API rehearsal request | Response code and body distinguish endpoint, authorization, object, and request-shape failures |
| Check authorization scope | `az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"` | Role scope is narrow enough and sufficient for the runtime path |

| Find application evidence | Application Insights > Transaction search > filter by operation id | Telemetry shows whether the dependency call happened and how it ended |
| Re-test original symptom | Repeat the original user action, queue message, event delivery, or API call | The same observable failure is gone after the targeted correction |

Store and Secure Source Documents in Azure Blob Storage

Container access level, managed identity data access, blob metadata, and private document ingestion path

Exam Radar

Core Priority: This topic belongs to "Develop AI solutions using Azure data management services" and focuses on the working object behind the service name: source document container.

High Frequency: A likely stem describes documents are uploaded but the AI pipeline cannot read or safely ingest them. The exam rewards evidence from source document container, not broad configuration changes.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while blob container access level, identity role, metadata convention, network path, and indexer data source is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when blob container access level, identity role, metadata convention, network path, and indexer data source does not match the workload execution path.

Operational Dependency: The workload depends on blob container access level, identity role, metadata convention, network path, and indexer data source. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: blob container access level, identity role, metadata convention, network path, and indexer data source. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

Blob Storage often holds the original documents before extraction, chunking, indexing, or analysis. The ingestion path needs access to private files without turning the container into anonymous public storage.

Metadata and permissions work together. Metadata helps downstream indexing classify documents; identity-based access keeps the source material private.

A practical learner should turn this topic into three questions: what selects source document container, what permission or route lets the app use it, and what evidence shows the call succeeded.

This sequence also keeps the learner from jumping to expensive fixes such as scaling, redeploying, or broadening permissions before the failed condition is known.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Container access level | Blob visibility | Private, blob, container | Private is safest baseline | Identity or SAS for ingestion | Sensitive source files become public |

| Blob metadata | Indexing hints | Content type, source id, tenant id, classification | Absent unless assigned | Indexer mapping and downstream filters | Documents index without governance context |
| Ingestion path | Blob source, indexer run, skillset output, or write operation | Scheduled, manual, or SDK-driven | No freshness guarantee | Credentials, mappings, and source availability | New documents exist but are not searchable |

| Query or read evidence | Search result ids, RU charge, point read, selected fields, filters | Response-specific | Unknown until observed | Request body and API version | Model receives the wrong context or state lookup slows down |
| Security boundary | Data-plane role, storage access, private network, or admin key | Least-privilege role or key | Unsafe if made public | Managed identity and network path | Private data exposure or 403 during ingestion |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is source document container; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.
2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az storage container show --account-name ai200docs --name source-docs --auth-mode login --query "

{name:name,publicAccess:properties.publicAccess}"
Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended source document container and the service-specific attributes connected to blob container access level, identity role, metadata convention, network path, and indexer data source.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   GET https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/rg-ai200/providers/Microsoft.Storage/storageAccounts/ai200docs/blobServices/default/containers/source-docs?api-version=2023-01-01
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.
4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.
5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

At runtime, code does not consume a service name; it consumes a configured object. For Azure Blob Storage, the request has to reach source document container, pass access checks, match the expected contract, and leave evidence in logs or status output.

When blob container access level, identity role, metadata convention, network path, and indexer data source is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |

| Inspect source document container | Run the Step 2 Azure CLI verification command | Output exposes the service state related to blob container access level, identity role, metadata convention, network path, and indexer data source |

| Confirm service/API behavior | Run the Step 3 REST/API rehearsal request | Response code and body distinguish endpoint, authorization, object, and request-shape failures |
| Check authorization scope | `az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"` | Role scope is narrow enough and sufficient for the runtime path |

| Find application evidence | Application Insights > Transaction search > filter by operation id | Telemetry shows whether the dependency call happened and how it ended |
| Re-test original symptom | Repeat the original user action, queue message, event delivery, or API call | The same observable failure is gone after the targeted correction |

Practice Questions

1. A RAG application returns no vector matches after embeddings were upgraded to a model that outputs a different dimension. Which configuration should be inspected first?
   A. Container Apps traffic split.
   B. Service Bus lock duration.
   C. Key Vault firewall mode.
   D. The Azure AI Search vector field dimension and vector search profile.

2. An Azure AI Search indexer completes, but documents cannot be retrieved by their expected IDs. Which mapping is the most likely first place to inspect?
   A. Document key preservation and output field mapping.
   B. Container Apps minimum replicas.
   C. Azure OpenAI temperature.
   D. Application Insights operation id.

3. A hybrid search query returns documents, but the answer is not grounded in the expected source material. What should be validated first?
   A. Whether the Service Bus queue is partitioned.
   B. The selected vector field, filter expression, top-k value, and returned document IDs.
   C. Whether the container image uses a slim base image.
   D. The Key Vault secret expiration date.

4. A conversation history lookup in Cosmos DB is slow and shows high request charge for a single user session. Which design element should be reviewed first?
   A. Azure Container Registry SKU.
   B. The Event Grid subject filter.
   C. Partition key design and whether point reads are possible.
   D. The Azure AI Search semantic configuration.

5. A document ingestion pipeline should read private source documents from Blob Storage using a managed identity. Which evidence confirms the data path is secure and reachable?
   A. A new Service Bus dead-letter rule.
   B. The Azure OpenAI max tokens value.
   C. A higher Container Apps max replica count.
   D. Blob container access level, managed identity data role, blob metadata, and private endpoint path.

6. An indexer run reports warnings about skillset output mappings, and expected enriched fields are empty. What should the developer check first?
   A. The Container Apps revision suffix.
   B. The skillset output field mappings and index schema field names.
   C. The ACR admin user setting.
   D. The Cosmos DB consistency level only.

7. A filter added to an Azure AI Search query removes all otherwise relevant RAG documents. Which action is most appropriate?
   A. Grant AcrPull to the search service.
   B. Restart the Container Apps environment.
   C. Inspect the filterable field values and compare returned document IDs with and without the filter.
   D. Move secrets to a different Key Vault.

8. A Cosmos DB query for one conversation scans multiple partitions. Which application behavior most likely caused this?
   A. The Event Grid delivery schema is CloudEvents.

B. The Azure AI Search indexer uses a schedule.
C. The model endpoint returns HTTP 429.
D. The query omits the partition key value needed for a point read or single-partition query.

9. A learner must compare searchable text fields and vector fields in an Azure AI Search index. Which distinction is most important for AI-200 scenarios?
A. Searchable text fields support lexical and semantic text retrieval; vector fields store embeddings with fixed dimensions for similarity search.
B. Vector fields store Key Vault secrets; searchable fields store Service Bus locks.
C. Searchable fields control Container Apps ingress; vector fields control revision traffic.
D. Both field types are interchangeable if the same analyzer is used.

10. A Blob Storage container is accidentally set to allow anonymous access for source documents. Which AI workload concern does this most directly affect?
A. Container Apps scale rule authentication.
B. Azure AI Search vector math.
C. Source document confidentiality and controlled ingestion path.
D. Application Insights dependency correlation.

Connect to and consume Azure services

Endpoint variables, client lifetime, retry policy, timeout settings, and API version selection

Exam Radar

Core Priority: This topic belongs to "Connect to and consume Azure services" and focuses on the working object behind the service name: service client configuration.

High Frequency: When the stem says the application intermittently fails across service calls with inconsistent endpoint and retry behavior, read it as an object-state problem first and a platform-change problem second.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while endpoint configuration, credential object, retry policy, timeout, and client reuse is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when endpoint configuration, credential object, retry policy, timeout, and client reuse does not match the workload execution path.

Operational Dependency: The workload depends on endpoint configuration, credential object, retry policy, timeout, and client reuse. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: endpoint configuration, credential object, retry policy, timeout, and client reuse. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

An Azure SDK client packages endpoint, credential, retry, timeout, serialization, and API version behavior. A poorly built client can make a healthy service look unreliable.

Client lifetime is a practical clue. Recreating clients on every request wastes connections and hides retry behavior; a stable client makes endpoint and credential resolution easier to inspect.

Study this as a runtime story rather than a service definition. The app points at service client configuration, Azure evaluates endpoint configuration, credential object, retry policy, timeout, and client reuse, and the result shows up as a status, log, metric, or response.

Once the object and access path are clear, the rest of the evidence has a place to attach: logs explain the call, metrics show pressure, and responses classify the failure.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Client endpoint | Service URL | Resource endpoint, regional endpoint, private endpoint | Read from environment or config | Correct resource and API route | SDK calls a development or wrong-region service |

| Retry policy | Transient failure handling | Exponential retry, fixed retry, disabled retry | SDK default varies by package | Timeout and service throttling behavior | Transient 429/503 becomes user-visible failure |
| Delivery or retry behavior | Lock duration, delivery count, retry policy, timeout, or Retry-After | Policy-bound values | Defaults may not fit AI latency | Worker duration and service throttling | Duplicate jobs, dead-letter growth, or user timeouts |

| Authentication input | Managed identity, key, token audience, or external secret | Provider-specific | Local and cloud may differ | Role assignment, Key Vault, and credential chain | 401/403 or accidental secret exposure |
| Integration evidence | Dependency telemetry, queue counts, event delivery metrics, or API response | Observable per service | Unavailable without logging or query | Application Insights and service metrics | Root cause is hidden behind a generic app exception |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is service client configuration; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.
2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az account get-access-token --resource https://management.azure.com/ --query "{tenant:tenant,expiresOn:expiresOn}"
   Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended service client configuration and the service-specific attributes connected to endpoint configuration, credential object, retry policy, timeout, and client reuse.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   GET https://management.azure.com/subscriptions/{subscriptionId}/resources?api-version=2021-04-01
   Authorization: Bearer
   Content-Type: application/json

Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.

4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.
5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

The execution chain is concrete: configuration selects service client configuration, identity or key proves access, networking reaches the endpoint, and the service validates the request against its current state.

When endpoint configuration, credential object, retry policy, timeout, and client reuse is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Inspect service client configuration | Run the Step 2 Azure CLI verification command | Output exposes the service state related to endpoint configuration, credential object, retry policy, timeout, and client reuse |

| Confirm service/API behavior | Run the Step 3 REST/API rehearsal request | Response code and body distinguish endpoint, authorization, object, and request-shape failures |
| Check authorization scope | `az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"` | Role scope is narrow enough and sufficient for the runtime path |

| Find application evidence | Application Insights > Transaction search > filter by operation id | Telemetry shows whether the dependency call happened and how it ended |
| Re-test original symptom | Repeat the original user action, queue message, event delivery, or API call | The same observable failure is gone after the targeted correction |

Consume Azure OpenAI or Azure AI Services Endpoints

Endpoint URL, deployment name, API version, request body, throttling response, and model-call validation

Exam Radar

Core Priority: This topic belongs to "Connect to and consume Azure services" and focuses on the working object behind the service name: model deployment endpoint.

High Frequency: This topic often appears as a small production incident: the AI call returns 404, 429, or a payload validation error while the resource exists. The useful option is the one that proves the next dependency in the chain.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while resource endpoint, deployment name, API version, request schema, rate limit, and identity or key authorization is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when resource endpoint, deployment name, API version, request schema, rate limit, and identity or key authorization does not match the workload execution path.

Operational Dependency: The workload depends on resource endpoint, deployment name, API version, request schema, rate limit, and identity or key authorization. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: resource endpoint, deployment name, API version, request schema, rate limit, and identity or key authorization. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

Azure AI service calls depend on resource endpoint, deployment or model identifier, API version, authorization, and request body. The resource can exist while the deployment name or route is still wrong.

Status codes give direction: 404 often means route or deployment mismatch, 401/403 means authentication or authorization, 429 means throttling, and 400 usually means request schema validation.

The compact mental model is: selected object, access path, accepted request, observable result. For this topic, all four revolve around model deployment endpoint.

The exam skill is choosing the first useful observation. A fix that happens before that observation is usually only a guess.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| --- | --- | --- | --- | --- | --- |
| Deployment name | Model route selector | Named deployment under resource | Not inferred from model family | Endpoint and API version | 404 despite a healthy Azure AI resource |

| Request body | Model operation contract | messages, input, max tokens, temperature, tool params | Invalid until service validates | API version and deployment capability | 400 schema or unsupported-parameter error |
| Delivery or retry behavior | Lock duration, delivery count, retry policy, timeout, or Retry-After | Policy-bound values | Defaults may not fit AI latency | Worker duration and service throttling | Duplicate jobs, dead-letter growth, or user timeouts |

| Authentication input | Managed identity, key, token audience, or external secret | Provider-specific | Local and cloud may differ | Role assignment, Key Vault, and credential chain | 401/403 or accidental secret exposure |
| Integration evidence | Dependency telemetry, queue counts, event delivery metrics, or API response | Observable per service | Unavailable without logging or query | Application Insights and service metrics | Root cause is hidden behind a generic app exception |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is model deployment endpoint; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.

2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az cognitiveservices account show --name ai200-openai --resource-group rg-ai200 --query "
   {endpoint:properties.endpoint,kind:kind,sku:sku.name}"
   Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended model deployment endpoint and the service-specific attributes connected to resource endpoint, deployment name, API version, request schema, rate limit, and identity or key authorization.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   POST https://ai200-openai.openai.azure.com/openai/deployments/{deploymentName}/chat/completions?api-version=2024-10-21
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.

4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.

5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

A user-visible result is the last link in the chain. Before that, Azure OpenAI or Azure AI Services resource plane has already evaluated the target object, the credential, the route, and the request contract.

When resource endpoint, deployment name, API version, request schema, rate limit, and identity or key authorization is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |

| Inspect model deployment endpoint | Run the Step 2 Azure CLI verification command | Output exposes the service state related to resource endpoint, deployment name, API version, request schema, rate limit, and identity or key authorization |

| Confirm service/API behavior | Run the Step 3 REST/API rehearsal request | Response code and body distinguish endpoint, authorization, object, and request-shape failures |
| Check authorization scope | `az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"` | Role scope is narrow enough and sufficient for the runtime path |

| Find application evidence | Application Insights > Transaction search > filter by operation id | Telemetry shows whether the dependency call happened and how it ended |
| Re-test original symptom | Repeat the original user action, queue message, event delivery, or API call | The same observable failure is gone after the targeted correction |

Send and Process Azure Service Bus Queue Messages

Message contract, lock duration, retry handling, dead-letter reason, and idempotent worker execution

Exam Radar

Core Priority: This topic belongs to "Connect to and consume Azure services" and focuses on the working object behind the service name: queue message.

High Frequency: Expect scenarios where AI enrichment jobs repeat or land in the dead-letter queue during model throttling. The best answer follows the failing runtime object, not the most visible Azure resource.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while message schema, lock duration, retry policy, dead-letter rule, and idempotency key is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when message schema, lock duration, retry policy, dead-letter rule, and idempotency key does not match the workload execution path.

Operational Dependency: The workload depends on message schema, lock duration, retry policy, dead-letter rule, and idempotency key. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: message schema, lock duration, retry policy, dead-letter rule, and idempotency key. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

Service Bus queues are for durable command-style work. A message is locked while a worker processes it; if processing exceeds the lock or fails repeatedly, it can be retried and eventually dead-lettered.

AI workers need idempotency because embedding, enrichment, and model calls can be retried. The worker should recognize a repeated job before writing duplicate results.

For hands-on study, begin with queue message: how it is named, how the app reaches it, and which field or status proves it is usable.

That order prevents cargo-cult troubleshooting. The command matters because it explains the symptom, not because it is a line to memorize.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Message lock | Exclusive processing window | ISO duration lock value | Queue default unless changed | Worker processing time | Message redelivers while worker is still running |

| Dead-letter reason | Terminal failure evidence | Max delivery, validation failure, explicit dead-letter | Empty until failure | Message schema and worker exception handling | Backlog diagnosis loses the real failure cause |
| Delivery or retry behavior | Lock duration, delivery count, retry policy, timeout, or Retry-After | Policy-bound values | Defaults may not fit AI latency | Worker duration and service throttling | Duplicate jobs, dead-letter growth, or user timeouts |

| Authentication input | Managed identity, key, token audience, or external secret | Provider-specific | Local and cloud may differ | Role assignment, Key Vault, and credential chain | 401/403 or accidental secret exposure |
| Integration evidence | Dependency telemetry, queue counts, event delivery metrics, or API response | Observable per service | Unavailable without logging or query | Application Insights and service metrics | Root cause is hidden behind a generic app exception |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is queue message; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.

2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az servicebus queue show --namespace-name ai200-bus --resource-group rg-ai200 --name embedding-jobs --query "{active:countDetails.activeMessageCount,deadLetter:countDetails.deadLetterMessageCount,lockDuration:lockDuration}"
   Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended queue message and the service-specific attributes connected to message schema, lock duration, retry policy, dead-letter rule, and idempotency key.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   GET https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/rg-ai200/providers/Microsoft.ServiceBus/namespaces/ai200-bus/queues/embedding-jobs?api-version=2024-01-01
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.

4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.

5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

The workload reaches Azure Service Bus by reading configuration, choosing credentials, resolving the endpoint, and sending a request to queue message. The service then checks authorization, object state, and request shape before it returns data or rejects

the operation.

When message schema, lock duration, retry policy, dead-letter rule, and idempotency key is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Inspect queue backlog | `az servicebus queue show --namespace-name ai200-bus --resource-group rg-ai200 --name embedding-jobs --query "countDetails"` | Active and dead-letter counts show whether workers are keeping up |

| Inspect lock and delivery policy | `az servicebus queue show --namespace-name ai200-bus --resource-group rg-ai200 --name embedding-jobs --query "{lockDuration:lockDuration,maxDeliveryCount:maxDeliveryCount}"` | Lock duration and delivery count match expected AI job latency |
| Inspect dead-letter reason | Azure portal path: Service Bus queue > Dead-letter messages > message properties | Dead-letter reason identifies schema, timeout, or worker failure |

| Inspect worker dependency failures | Application Insights > dependencies filtered by operation id | Model or storage failures align with message processing attempts |
| Validate idempotency evidence | Application log query for job id or idempotency key | Repeated delivery does not create duplicate output records |

Handle Azure Event Grid Notifications for AI Pipelines

Event subscription filter, endpoint validation handshake, event schema, retry policy, and delivery metrics

Exam Radar

Core Priority: This topic belongs to "Connect to and consume Azure services" and focuses on the working object behind the service name: event subscription.

High Frequency: A likely stem describes document upload events do not start the AI ingestion workflow. The exam rewards evidence from event subscription, not broad configuration changes.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while event type, subject filter, endpoint validation, delivery destination, and retry metrics is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when event type, subject filter, endpoint validation, delivery destination, and retry metrics does not match the workload execution path.

Operational Dependency: The workload depends on event type, subject filter, endpoint validation, delivery destination, and retry metrics. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: event type, subject filter, endpoint validation, delivery destination, and retry metrics. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

Event Grid announces that something happened, such as a blob creation event. It is not a durable ordered work queue, so it should trigger a workflow rather than replace command processing.

Subscription filters decide which events are delivered. A healthy storage account does not prove that the event subscription matched the subject, validated the endpoint, or delivered the event.

A practical learner should turn this topic into three questions: what selects event subscription, what permission or route lets the app use it, and what evidence shows the call succeeded.

This sequence also keeps the learner from jumping to expensive fixes such as scaling, redeploying, or broadening permissions before the failed condition is known.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Subject filter | Event selection rule | Prefix/suffix and event type filters | Broad or absent unless configured | Source event schema | Uploads occur but matching events are never delivered |

| Endpoint validation | Subscriber handshake | Validation event and response | Required for many destinations | Webhook or handler implementation | Subscription never becomes ready |
| Delivery or retry behavior | Lock duration, delivery count, retry policy, timeout, or Retry-After | Policy-bound values | Defaults may not fit AI latency | Worker duration and service throttling | Duplicate jobs, dead-letter growth, or user timeouts |

| Authentication input | Managed identity, key, token audience, or external secret | Provider-specific | Local and cloud may differ | Role assignment, Key Vault, and credential chain | 401/403 or accidental secret exposure |
| Integration evidence | Dependency telemetry, queue counts, event delivery metrics, or API response | Observable per service | Unavailable without logging or query | Application Insights and service metrics | Root cause is hidden behind a generic app exception |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is event subscription; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.
2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az eventgrid event-subscription show --name ai200-blob-events --source-resource-id /subscriptions/{subscriptionId}/resourceGroups/rg-ai200/providers/Microsoft.Storage/storageAccounts/ai200docs --query "{state:provisioningState,filter:filter}"
   Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended event subscription and the service-specific attributes connected to event type, subject filter, endpoint validation, delivery destination, and retry metrics.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   GET https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/rg-ai200/providers/Microsoft.Storage/storageAccounts/ai200docs/providers/Microsoft.EventGrid/eventSubscriptions/ai200-blob-events?api-version=2023-12-15-preview
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.
4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.
5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

At runtime, code does not consume a service name; it consumes a configured object. For Azure Event Grid, the request has to reach event subscription, pass access checks, match the expected contract, and leave evidence in logs or status output.

When event type, subject filter, endpoint validation, delivery destination, and retry metrics is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

Task	Precise Command or Path	Verification Standard
Inspect event subscription	Run the Step 2 Azure CLI verification command	Output exposes the service state related to event type, subject filter, endpoint validation, delivery destination, and retry metrics
Confirm service/API behavior	Run the Step 3 REST/API rehearsal request	Response code and body distinguish endpoint, authorization, object, and request-shape failures
Check authorization scope	`az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"`	Role scope is narrow enough and sufficient for the runtime path
Find application evidence	Application Insights > Transaction search > filter by operation id	Telemetry shows whether the dependency call happened and how it ended
Re-test original symptom	Repeat the original user action, queue message, event delivery, or API call	The same observable failure is gone after the targeted correction

Call External APIs from an Azure AI Back End

HTTP client timeout, retry classification, secret retrieval, response contract, and failure isolation

Exam Radar

Core Priority: This topic belongs to "Connect to and consume Azure services" and focuses on the working object behind the service name: external API dependency.

High Frequency: When the stem says the AI workflow fails only when enriching responses with an external service, read it as an object-state problem first and a platform-change problem second.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while base URL, authentication secret, timeout, retry classification, response schema, and dependency telemetry is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when base URL, authentication secret, timeout, retry classification, response schema, and dependency telemetry does not match the workload execution path.

Operational Dependency: The workload depends on base URL, authentication secret, timeout, retry classification, response schema, and dependency telemetry. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: base URL, authentication secret, timeout, retry classification, response schema, and dependency telemetry. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

External APIs add failure modes outside Azure: DNS, TLS, authentication, rate limits, response shape changes, and timeout budgets. Those failures should appear as dependency telemetry, not as vague model failures.

The response contract matters because AI enrichment code often assumes fields from another system. A 200 response with a changed JSON shape can break the workflow as surely as a 500 response.

Study this as a runtime story rather than a service definition. The app points at external API dependency, Azure evaluates base URL, authentication secret, timeout, retry classification, response schema, and dependency telemetry, and the result shows up as a status, log, metric, or response.

Once the object and access path are clear, the rest of the evidence has a place to attach: logs explain the call, metrics show pressure, and responses classify the failure.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| HTTP timeout | Latency budget | Connection, read, total timeout | Client default may be too long | User request SLA and retry policy | Threads or async tasks wait until API route times out |

| Response contract | Expected JSON shape | Required fields, status code, content type | Assumed by code | External API version and schema | 200 response still breaks enrichment logic |
| Delivery or retry behavior | Lock duration, delivery count, retry policy, timeout, or Retry-After | Policy-bound values | Defaults may not fit AI latency | Worker duration and service throttling | Duplicate jobs, dead-letter growth, or user timeouts |

| Authentication input | Managed identity, key, token audience, or external secret | Provider-specific | Local and cloud may differ | Role assignment, Key Vault, and credential chain | 401/403 or accidental secret exposure |
| Integration evidence | Dependency telemetry, queue counts, event delivery metrics, or API response | Observable per service | Unavailable without logging or query | Application Insights and service metrics | Root cause is hidden behind a generic app exception |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is external API dependency; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.
2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az monitor app-insights query --app ai200-aiapi --analytics-query "dependencies | where type == 'HTTP' | summarize failures=countif(success == false), p95=percentile(duration,95) by target"
   Command note: The external API URL is intentionally a lab placeholder. In production, replace it with the real dependency and validate timeout, auth, and response schema through dependency telemetry.

Expected checkpoint: the output shows the intended external API dependency and the service-specific attributes connected to base URL, authentication secret, timeout, retry classification, response schema, and dependency telemetry.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   GET https://api.contoso.example/v1/resources/{id}
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.
4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.
5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

The execution chain is concrete: configuration selects external API dependency, identity or key proves access, networking reaches the endpoint, and the service validates the request against its current state.

When base URL, authentication secret, timeout, retry classification, response schema, and dependency telemetry is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

Task	Precise Command or Path	Verification Standard
Inspect external API dependency	Run the Step 2 Azure CLI verification command	Output exposes the service state related to base URL, authentication secret, timeout, retry classification, response schema, and dependency telemetry
Confirm service/API behavior	Run the Step 3 REST/API rehearsal request	Response code and body distinguish endpoint, authorization, object, and request-shape failures
Check authorization scope	`az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"`	Role scope is narrow enough and sufficient for the runtime path
Find application evidence	Application Insights > Transaction search > filter by operation id	Telemetry shows whether the dependency call happened and how it ended
Re-test original symptom	Repeat the original user action, queue message, event delivery, or API call	The same observable failure is gone after the targeted correction

Practice Questions

1. An AI application intermittently fails because a new SDK client is created for every request under high load. Which configuration area should be reviewed first?
   A. Blob Storage anonymous access level.
   B. Azure AI Search document key mapping.
   C. Container Apps traffic weight only.
   D. Client lifetime, retry policy, timeout settings, and endpoint variables.

2. A call to an Azure OpenAI deployment returns 404 even though the resource exists. What should be verified first?
   A. The Service Bus dead-letter reason.
   B. The Key Vault private endpoint DNS zone.
   C. Endpoint URL, deployment name, API version, and request path.
   D. The Container Apps minimum replica count.

3. A Service Bus-triggered AI worker processes the same message more than once after long model calls. Which setting should be inspected first?
   A. Azure AI Search semantic ranker.
   B. Message lock duration, renewal behavior, and idempotent processing logic.
   C. Azure Container Registry repository tags.
   D. Event Grid subject prefix.

4. An Event Grid subscription is created for an AI pipeline endpoint, but no production events are delivered. Which evidence should be checked first?
   A. Cosmos DB request charge.
   B. Container Apps CPU limit.
   C. Azure OpenAI response temperature.
   D. Endpoint validation handshake, subject filters, event schema, and delivery metrics.

5. An AI back end calls an external API and sometimes blocks request threads until users time out. What is the most appropriate first design check?
   A. HTTP client timeout, retry classification, response contract, and failure isolation.
   B. The vector field dimension in Azure AI Search.

C. The ACR image digest.
D. The Key Vault secret version count.

6. A Service Bus message repeatedly fails because the worker cannot parse the payload schema. Where should the developer look first?
   A. The Application Insights sampling rate.
   B. The message contract, version field, and dead-letter reason.
   C. Container Apps ingress target port.
   D. Azure AI Search top-k value.

7. A model call receives 429 responses during peak traffic. Which response detail should guide retry behavior first?
   A. The Dockerfile exposed port.
   B. The Event Grid validation code.
   C. Retry-After or service throttling metadata from the endpoint response.
   D. The Blob Storage container metadata.

8. A developer stores an external API key in configuration and wants to remove static secrets from the AI back end. Which dependency chain should be implemented?
   A. Increase the Service Bus max delivery count only.
   B. Enable public anonymous access on Blob Storage.
   C. Use an ACR tag instead of a digest.
   D. Use managed identity to retrieve the secret from Key Vault, then call the external API with controlled timeout and retry handling.

9. An Event Grid handler receives events but rejects them because the body shape is different from the code expectation. Which configuration is most relevant?
   A. The selected Event Grid event schema and handler deserialization contract.
   B. The Cosmos DB partition key.
   C. The Container Apps scale cooldown only.
   D. The Azure AI Search index analyzer.

10. An SDK-based AI workload sometimes hangs for a long time when an Azure dependency is unavailable. Which setting is most likely missing or too permissive?
    A. Service Bus queue name.
    B. Client timeout and bounded retry configuration.
    C. Search index semantic title field.
    D. ACR repository retention policy.

Secure, monitor, and troubleshoot Azure solutions

Identity assignment, token audience, RBAC scope, SDK credential chain, and runtime principal validation

Exam Radar

Core Priority: This topic belongs to "Secure, monitor, and troubleshoot Azure solutions" and focuses on the working object behind the service name: runtime identity.

High Frequency: This topic often appears as a small production incident: local development authentication works but the deployed AI service receives 401 or 403. The useful option is the one that proves the next dependency in the chain.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while assigned identity, token audience, role assignment scope, SDK credential order, and runtime environment is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when assigned identity, token audience, role assignment scope, SDK credential order, and runtime environment does not match the workload execution path.

Operational Dependency: The workload depends on assigned identity, token audience, role assignment scope, SDK credential order, and runtime environment. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: assigned identity, token audience, role assignment scope, SDK credential order, and runtime environment. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

Managed identity gives an Azure resource its own Microsoft Entra identity. The app requests a token for a specific audience, and the target service checks both the token audience and the role assignment.

Token audience explains many 401/403 cases. A token issued for the wrong resource is not accepted by the target service even when the identity exists and has some Azure role.

The compact mental model is: selected object, access path, accepted request, observable result. For this topic, all four revolve around runtime identity.

The exam skill is choosing the first useful observation. A fix that happens before that observation is usually only a guess.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Token audience | Resource identifier for token | Management, Key Vault, Cognitive Services, Storage, Search | Chosen by SDK or request | Target service authentication | 401/403 even with a valid identity |

| Principal id | Runtime identity object | System-assigned or user-assigned principal | Absent until identity assigned | RBAC role assignment | Local auth works while cloud runtime fails |
| Failure classifier | 401, 403, 404, 429, 503, timeout, exception type, or dead-letter reason | Service-specific meanings | Unknown until captured | Logs, dependencies, and response headers | Fix targets the symptom instead of the failing dependency |

| Correlation mechanism | operation_Id, request id, message id, correlation id, or timestamp | Generated per request or workflow | Lost if not propagated | Instrumentation and logging conventions | Timeline cannot identify the first failing link |
| Validation output | KQL result, metric chart, secret response, token claim, or retry evidence | Measured evidence | Untrusted until filtered | Azure Monitor, App Insights, Key Vault, and service logs | Incident response relies on a dashboard guess |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is runtime identity; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.

2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az containerapp identity show --name ai-api --resource-group rg-ai200 --query "
   {principalId:principalId,tenantId:tenantId,type:type}"
   Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended runtime identity and the service-specific attributes connected to assigned identity, token audience, role assignment scope, SDK credential order, and runtime environment.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   GET https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/rg-ai200/providers/Microsoft.App/containerApps/ai-api?api-version=2024-03-01
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.

4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.

5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

A user-visible result is the last link in the chain. Before that, Microsoft Entra managed identity has already evaluated the target object, the credential, the route, and the request contract.

When assigned identity, token audience, role assignment scope, SDK credential order, and runtime environment is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| --- | --- | --- |

| Inspect runtime identity | Run the Step 2 Azure CLI verification command | Output exposes the service state related to assigned identity, token audience, role assignment scope, SDK credential order, and runtime environment |

| Confirm service/API behavior | Run the Step 3 REST/API rehearsal request | Response code and body distinguish endpoint, authorization, object, and request-shape failures |
| Check authorization scope | `az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"` | Role scope is narrow enough and sufficient for the runtime path |

| Find application evidence | Application Insights > Transaction search > filter by operation id | Telemetry shows whether the dependency call happened and how it ended |
| Re-test original symptom | Repeat the original user action, queue message, event delivery, or API call | The same observable failure is gone after the targeted correction |

Retrieve Secrets from Azure Key Vault in AI Workloads

Vault RBAC, secret URI, firewall path, private endpoint DNS, and secret client response validation

Exam Radar

Core Priority: This topic belongs to "Secure, monitor, and troubleshoot Azure solutions" and focuses on the working object behind the service name: secret retrieval request.

High Frequency: Expect scenarios where the service cannot retrieve endpoint or key configuration during startup. The best answer follows the failing runtime object, not the most visible Azure resource.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while secret name, vault access model, role assignment, firewall rule, private DNS, and SDK identity is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when secret name, vault access model, role assignment, firewall rule, private DNS, and SDK identity does not match the workload execution path.

Operational Dependency: The workload depends on secret name, vault access model, role assignment, firewall rule, private DNS, and SDK identity. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: secret name, vault access model, role assignment, firewall rule, private DNS, and SDK identity. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

Key Vault secret retrieval requires the secret identifier, permission to read it, and network reachability to the vault endpoint. Missing each one creates a different failure pattern.

For private vaults, DNS is part of the security design. The app may have the right role but still fail if the vault name resolves incorrectly from the runtime subnet.

For hands-on study, begin with secret retrieval request: how it is named, how the app reaches it, and which field or status proves it is usable.

That order prevents cargo-cult troubleshooting. The command matters because it explains the symptom, not because it is a line to memorize.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Secret identifier | Vault URI and secret name | https://vault.vault.azure.net/secrets/name/version | Unknown until configured | Secret client and app setting | Startup cannot find the endpoint or key value |

| Vault firewall | Network access policy | Public, selected networks, private endpoint | Policy-dependent | Runtime subnet and DNS | Correct role still fails with network denial |
| Failure classifier | 401, 403, 404, 429, 503, timeout, exception type, or dead-letter reason | Service-specific meanings | Unknown until captured | Logs, dependencies, and response headers | Fix targets the symptom instead of the failing dependency |

| Correlation mechanism | operation_Id, request id, message id, correlation id, or timestamp | Generated per request or workflow | Lost if not propagated | Instrumentation and logging conventions | Timeline cannot identify the first failing link |
| Validation output | KQL result, metric chart, secret response, token claim, or retry evidence | Measured evidence | Untrusted until filtered | Azure Monitor, App Insights, Key Vault, and service logs | Incident response relies on a dashboard guess |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is secret retrieval request; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.
2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az keyvault secret show --vault-name ai200-kv --name openai-endpoint --query "{id:id,enabled:attributes.enabled}"
   Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended secret retrieval request and the service-specific attributes connected to secret name, vault access model, role assignment, firewall rule, private DNS, and SDK identity.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   GET https://ai200-kv.vault.azure.net/secrets/openai-endpoint?api-version=7.4
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.
4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.
5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

The workload reaches Azure Key Vault by reading configuration, choosing credentials, resolving the endpoint, and sending a request to secret retrieval request. The service then checks authorization, object state, and request shape before it returns data or rejects the operation.

When secret name, vault access model, role assignment, firewall rule, private DNS, and SDK identity is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Inspect secret retrieval request | Run the Step 2 Azure CLI verification command | Output exposes the service state related to secret name, vault access model, role assignment, firewall rule, private DNS, and SDK identity |

| Confirm service/API behavior | Run the Step 3 REST/API rehearsal request | Response code and body distinguish endpoint, authorization, object, and request-shape failures |
| Check authorization scope | `az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"` | Role scope is narrow enough and sufficient for the runtime path |

| Find application evidence | Application Insights > Transaction search > filter by operation id | Telemetry shows whether the dependency call happened and how it ended |
| Re-test original symptom | Repeat the original user action, queue message, event delivery, or API call | The same observable failure is gone after the targeted correction |

Instrument AI APIs with Application Insights

Request telemetry, dependency telemetry, exception capture, trace correlation, and operation id analysis

Exam Radar

Core Priority: This topic belongs to "Secure, monitor, and troubleshoot Azure solutions" and focuses on the working object behind the service name: distributed trace.

High Frequency: A likely stem describes users report slow AI answers but infrastructure metrics look normal. The exam rewards evidence from distributed trace, not broad configuration changes.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while instrumentation connection, operation id, dependency tracking, exception logging, and latency percentile query is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when instrumentation connection, operation id, dependency tracking, exception logging, and latency percentile query does not match the workload execution path.

Operational Dependency: The workload depends on instrumentation connection, operation id, dependency tracking, exception logging, and latency percentile query. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: instrumentation connection, operation id, dependency tracking, exception logging, and latency percentile query. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

`requests` show inbound calls to the app, `dependencies` show outbound calls to services, `exceptions` show captured failures, and `traces` show application-written diagnostic messages.

Operation id ties those records together. Without correlation, a learner sees many facts but cannot prove which dependency failed for the user request in the scenario.

A practical learner should turn this topic into three questions: what selects distributed trace, what permission or route lets the app use it, and what evidence shows the call succeeded.

This sequence also keeps the learner from jumping to expensive fixes such as scaling, redeploying, or broadening permissions before the failed condition is known.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| requests table | Inbound operation record | Name, resultCode, duration, success | Missing without instrumentation | App SDK and connection string | User-facing failure cannot be measured |

| dependencies table | Outbound service call | Target, type, resultCode, duration | Missing if dependency tracking disabled | SDK instrumentation and operation context | Search/model/Cosmos failures are invisible |
| Failure classifier | 401, 403, 404, 429, 503, timeout, exception type, or dead-letter reason | Service-specific meanings | Unknown until captured | Logs, dependencies, and response headers | Fix targets the symptom instead of the failing dependency |

| Correlation mechanism | operation_Id, request id, message id, correlation id, or timestamp | Generated per request or workflow | Lost if not propagated | Instrumentation and logging conventions | Timeline cannot identify the first failing link |
| Validation output | KQL result, metric chart, secret response, token claim, or retry evidence | Measured evidence | Untrusted until filtered | Azure Monitor, App Insights, Key Vault, and service logs | Incident response relies on a dashboard guess |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is distributed trace; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.
2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az monitor app-insights query --app ai200-aiapi --analytics-query "requests | summarize failures=countif(success == false), p95=percentile(duration,95) by cloud_RoleName"
   Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended distributed trace and the service-specific attributes connected to instrumentation connection, operation id, dependency tracking, exception logging, and latency percentile query.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   POST https://api.applicationinsights.io/v1/apps/{appId}/query
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.
4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.
5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

At runtime, code does not consume a service name; it consumes a configured object. For Azure Monitor Application Insights, the request has to reach distributed trace, pass access checks, match the expected contract, and leave evidence in logs or status output.

When instrumentation connection, operation id, dependency tracking, exception logging, and latency percentile query is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Inspect inbound requests | Application Insights Logs: run request failure and p95 latency query | Shows whether the user-facing API route is failing or slow |

| Inspect outbound dependencies | Application Insights Logs: run dependency result-code query by target | Identifies failing calls to Search, model endpoints, Cosmos DB, or external APIs |
| Inspect exceptions | Application Insights Logs: group exceptions by type and message | Shows code-level failures attached to the same operation |

| Inspect traces | Application Insights Logs: filter traces by operation id and order by timestamp | Explains application decisions between request and dependency call |
| Validate transaction correlation | Application Insights > Transaction search > select operation id | Requests, dependencies, exceptions, and traces appear in one timeline |

Troubleshoot AI Service Throttling and Retry Failures

HTTP status classification, Retry-After handling, SDK retry policy, queue backpressure, and user-facing timeout control

Exam Radar

Core Priority: This topic belongs to "Secure, monitor, and troubleshoot Azure solutions" and focuses on the working object behind the service name: throttled model request.

High Frequency: When the stem says AI requests fail during traffic spikes and queued jobs begin to age, read it as an object-state problem first and a platform-change problem second.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while status code, Retry-After header, SDK retry policy, concurrency, queue backpressure, and timeout budget is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when status code, Retry-After header, SDK retry policy, concurrency, queue backpressure, and timeout budget does not match the workload execution path.

Operational Dependency: The workload depends on status code, Retry-After header, SDK retry policy, concurrency, queue backpressure, and timeout budget. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: status code, Retry-After header, SDK retry policy, concurrency, queue backpressure, and timeout budget. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

Throttling is a service protection signal, not a random failure. Status codes such as 429 or 503 and headers such as Retry-After tell the client how quickly to try again.

Retries must fit the workload. Interactive API calls need bounded latency; queued background jobs can tolerate delayed retry and backpressure.

Study this as a runtime story rather than a service definition. The app points at throttled model request, Azure evaluates status code, Retry-After header, SDK retry policy, concurrency, queue backpressure, and timeout budget, and the result shows up as a status, log, metric, or response.

Once the object and access path are clear, the rest of the evidence has a place to attach: logs explain the call, metrics show pressure, and responses classify the failure.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Retry-After header | Backoff instruction | Seconds or date-based delay | Present only on some throttles | Client retry policy | Client retries too quickly and amplifies throttling |

| Dependency result code | Throttling classifier | 429, 503, timeout, gateway error | Unknown until logged | Application Insights dependency telemetry | Capacity issue is mistaken for application bug |
| Failure classifier | 401, 403, 404, 429, 503, timeout, exception type, or dead-letter reason | Service-specific meanings | Unknown until captured | Logs, dependencies, and response headers | Fix targets the symptom instead of the failing dependency |

| Correlation mechanism | operation_Id, request id, message id, correlation id, or timestamp | Generated per request or workflow | Lost if not propagated | Instrumentation and logging conventions | Timeline cannot identify the first failing link |
| Validation output | KQL result, metric chart, secret response, token claim, or retry evidence | Measured evidence | Untrusted until filtered | Azure Monitor, App Insights, Key Vault, and service logs | Incident response relies on a dashboard guess |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is throttled model request; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.

2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az monitor app-insights query --app ai200-aiapi --analytics-query "dependencies | where resultCode in ('429','503') | summarize count(), p95=percentile(duration,95) by target, resultCode"
   Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended throttled model request and the service-specific attributes connected to status code, Retry-After header, SDK retry policy, concurrency, queue backpressure, and timeout budget.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   POST https://ai200-openai.openai.azure.com/openai/deployments/{deploymentName}/chat/completions?api-version=2024-10-21
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.

4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.

5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

The execution chain is concrete: configuration selects throttled model request, identity or key proves access, networking reaches the endpoint, and the service validates the request against its current state.

When status code, Retry-After header, SDK retry policy, concurrency, queue backpressure, and timeout budget is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Inspect throttled model request | Run the Step 2 Azure CLI verification command | Output exposes the service state related to status code, Retry-After header, SDK retry policy, concurrency, queue backpressure, and timeout budget |

| Confirm service/API behavior | Run the Step 3 REST/API rehearsal request | Response code and body distinguish endpoint, authorization, object, and request-shape failures |
| Check authorization scope | `az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"` | Role scope is narrow enough and sufficient for the runtime path |

| Find application evidence | Application Insights > Transaction search > filter by operation id | Telemetry shows whether the dependency call happened and how it ended |
| Re-test original symptom | Repeat the original user action, queue message, event delivery, or API call | The same observable failure is gone after the targeted correction |

Audit Logs and Metrics for Azure AI Solution Incidents

Log source selection, metric dimension filtering, alert evidence, and root-cause timeline reconstruction

Exam Radar

Core Priority: This topic belongs to "Secure, monitor, and troubleshoot Azure solutions" and focuses on the working object behind the service name: incident evidence timeline.

High Frequency: This topic often appears as a small production incident: multiple Azure services show warnings and the developer must isolate the first failing dependency. The useful option is the one that proves the next dependency in the chain.

Confusion Alert: Resource existence does not prove runtime success. The exam often describes a deployed service while metric namespace, log table, time range, dimension filter, alert rule, and correlation id is still wrong for the path the application actually uses.

Scenario Logic: Read the stem as a chain: caller, configuration, credential, endpoint, service object, response, and telemetry. The useful clue is the first link where the chain can be observed.

Version Delta: AI-200 is beta. Stable Azure CLI patterns are included where useful; REST examples are rehearsal patterns and should be checked against current Microsoft Learn API documentation before live use.

Failure Trigger: The failure appears when metric namespace, log table, time range, dimension filter, alert rule, and correlation id does not match the workload execution path.

Operational Dependency: The workload depends on metric namespace, log table, time range, dimension filter, alert rule, and correlation id. If that dependency is wrong, a correct-looking architecture still fails.

How the Exam Asks It: Expect wording such as first step, best way to verify, least privilege, minimal change, troubleshoot, or which configuration resolves the symptom.

How Distractors Are Designed: Wrong answers are often useful actions in the wrong order: rebuilds before state checks, scaling before backlog evidence, broader permissions before identity proof, or prompt tuning before retrieval evidence.

Why the Correct Answer Works: The right answer proves the next required condition in the workflow: metric namespace, log table, time range, dimension filter, alert rule, and correlation id. It narrows the problem instead of making a broad platform change.

Atomic Deconstruction - Operational Level

Incident analysis reconstructs a timeline from metrics, logs, alerts, and correlation ids. The first failing dependency is often earlier than the loudest symptom.

Metrics show numeric behavior over time; logs explain individual events. Strong troubleshooting combines both instead of trusting one dashboard tile.

The compact mental model is: selected object, access path, accepted request, observable result. For this topic, all four revolve around incident evidence timeline.

The exam skill is choosing the first useful observation. A fix that happens before that observation is usually only a guess.

Component Specifications

| Object | Attribute | Value Range | Default State | Dependency | Failure State |

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Metric dimension | Filtered numeric signal | Status code, revision, target, namespace, operation | Broad until filtered | Resource metric namespace | Dashboard average hides failing slice |

| Correlation id | Timeline join key | operation id, request id, message id, timestamp | Lost if not propagated | Logging convention and telemetry context | Incident timeline cannot identify first failure |
| Failure classifier | 401, 403, 404, 429, 503, timeout, exception type, or dead-letter reason | Service-specific meanings | Unknown until captured | Logs, dependencies, and response headers | Fix targets the symptom instead of the failing dependency |

| Correlation mechanism | operation_Id, request id, message id, correlation id, or timestamp | Generated per request or workflow | Lost if not propagated | Instrumentation and logging conventions | Timeline cannot identify the first failing link |
| Validation output | KQL result, metric chart, secret response, token claim, or retry evidence | Measured evidence | Untrusted until filtered | Azure Monitor, App Insights, Key Vault, and service logs | Incident response relies on a dashboard guess |

Step-by-Step Execution Path

1. Start from the symptom and write the object name the application is actually using. For this topic, that object is incident evidence timeline; find it in configuration, deployment output, SDK client construction, message metadata, or the Azure portal resource blade.
2. Validate the Azure-side state. Command type: official Azure CLI verification pattern.
   az monitor metrics list --resource /subscriptions/{subscriptionId}/resourceGroups/rg-ai200/providers/Microsoft.App/containerApps/ai-api --metric Requests --interval PT5M
   Command note: This command is written as an official Azure CLI verification pattern. Confirm installed extension versions and optional JMESPath fields in the active lab environment.

Expected checkpoint: the output shows the intended incident evidence timeline and the service-specific attributes connected to metric namespace, log table, time range, dimension filter, alert rule, and correlation id.

3. Validate the service behavior from the request side. Command type: REST/API rehearsal; confirm the active API version and authorization method before production use.
   POST https://api.loganalytics.io/v1/workspaces/{workspaceId}/query
   Authorization: Bearer
   Content-Type: application/json
   Expected checkpoint: the status code and body distinguish name mismatch, authorization failure, request schema failure, throttling, and service-side processing errors.
4. Check the evidence source that belongs to this service: revision status for Container Apps, indexer status for AI Search, request charge for Cosmos DB, queue counts for Service Bus, delivery metrics for Event Grid, or operation-id telemetry for Application Insights.
5. Change only the broken dependency and repeat the same observation. The original failure should disappear because the inspected state changed, not because unrelated configuration drift masked the symptom.

Technical Chain

A user-visible result is the last link in the chain. Before that, Azure Monitor logs and metrics has already evaluated the target object, the credential, the route, and the request contract.

When metric namespace, log table, time range, dimension filter, alert rule, and correlation id is wrong, the failure often appears one layer later as a timeout, 401/403, 404, 400, stale result, retry storm, or generic application exception. The exam answer is strongest when it names the earliest observable link and uses that evidence to decide the next action.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Inspect incident evidence timeline | Run the Step 2 Azure CLI verification command | Output exposes the service state related to metric namespace, log table, time range, dimension filter, alert rule, and correlation id |

| Confirm service/API behavior | Run the Step 3 REST/API rehearsal request | Response code and body distinguish endpoint, authorization, object, and request-shape failures |
| Check authorization scope | `az role assignment list --assignee <principalId> --all --query "[].{role:roleDefinitionName,scope:scope}"` | Role scope is narrow enough and sufficient for the runtime path |

| Find application evidence | Application Insights > Transaction search > filter by operation id | Telemetry shows whether the dependency call happened and how it ended |
| Re-test original symptom | Repeat the original user action, queue message, event delivery, or API call | The same observable failure is gone after the targeted correction |

Practice Questions

1. A managed identity-enabled AI API receives 403 when reading from Azure AI Search. What should be validated first?
   A. Application Insights chart color.
   B. The image base layer size.
   C. The Event Grid retry schedule.
   D. Token audience, runtime principal, and data-plane role assignment scope.

2. An AI workload cannot retrieve a secret from Key Vault over a private endpoint and receives timeouts rather than authorization errors. What should be checked first?
   A. Key Vault private endpoint DNS, firewall path, and network reachability.
   B. Azure AI Search top-k.
   C. Service Bus dead-letter count.
   D. Container Apps traffic label.

3. A user request fails after a downstream dependency returns an error. Which Application Insights evidence best correlates the user-facing request with the dependency failure?
   A. The ACR repository name.
   B. The operation id across request, dependency, exception, and trace telemetry.
   C. The Cosmos DB container throughput setting only.
   D. The Event Grid subject suffix.

4. An AI API returns 429 responses from a model endpoint, and queue backlog grows at the same time. What should the team inspect before increasing compute?
   A. The Dockerfile base image digest only.
   B. The Blob Storage public access flag only.

C. The Retry-After value, SDK retry policy, dependency telemetry, and queue backpressure.
D. The Event Grid validation handshake only.

5. An incident review needs to determine whether failures started after a new revision, a Key Vault access change, or a dependency outage. Which artifact is most useful?
   A. A new ACR tag without checking the digest.
   B. A new Azure AI Search synonym map.
   C. A higher max token setting.
   D. A root-cause timeline built from deployment events, logs, metrics, alerts, and dependency telemetry.
6. A managed identity token is acquired successfully, but the target Azure service still rejects the request. Which distinction should be checked first?
   A. Whether the token audience and RBAC assignment match the target service and operation.
   B. Whether the Dockerfile uses multi-stage build.
   C. Whether Event Grid uses batching.
   D. Whether the queue has duplicate detection enabled.
7. A Key Vault secret URI in configuration points to a deleted or wrong secret name. Which symptom is most likely?
   A. Container Apps will always route traffic to the old revision.
   B. Azure AI Search vector dimensions automatically change.
   C. The secret client returns a not-found or retrieval error for the configured URI.
   D. Service Bus locks renew indefinitely.
8. A team sees only successful AI API requests in logs but users report failures during downstream calls. What telemetry gap should be investigated?
   A. Azure Container Registry webhook payloads.
   B. Missing dependency and exception telemetry correlated to the request operation.
   C. Blob index tags only.
   D. The Event Grid subject filter only.
9. An alert fires for increased dependency duration on an external API used by the AI back end. Which metric dimension or evidence is most useful for first triage?
   A. The Service Bus queue authorization rule name only.
   B. The Azure AI Search analyzer language.
   C. The ACR repository description.
   D. The dependency target, result code, duration trend, and affected operation ids.
10. An AI API uses managed identity locally during development and in Container Apps at runtime. What should be documented to avoid environment-specific credential confusion?
    A. The SDK credential chain, selected principal in each environment, and validation evidence for the runtime principal.
    B. Only the developer's local Azure CLI subscription name.
    C. Only the Container Apps environment region.
    D. Only the Azure AI Search index analyzer.

Learning Path & Study Advice

- Start with the Knowledge Overview so you can see the full exam scope and the exact order of the official domains, beginning with Develop containerized solutions on Azure, Develop AI solutions using Azure data management services, Connect to and consume Azure services.

- Read the Core Explanation in each knowledge point first to build a clean baseline understanding of the terminology, technologies, and customer scenarios.
- Continue into the Advanced Explanation to deepen your understanding of design trade-offs, deployment planning, optimization options, and operational decision-making.
- Work through the Practice Questions immediately after each knowledge point and answer them before checking the attachment section to strengthen retention.
- Revisit the answer attachment to identify weak areas, then loop back into the corresponding knowledge-point section for targeted review.

Who This PDF Is For

This study pack is intended for learners preparing for the Microsoft Certified: Azure AI Cloud Developer Associate (beta) exam who want a structured, exam-aligned review resource. It is especially useful for AI engineers, cloud developers, solution architects, data professionals, and Azure practitioners who need to connect Azure AI services with practical implementation and solution design tasks.

It is also a good fit for self-paced learners who prefer to study from organized knowledge points, detailed explanations, and directly paired practice questions instead of jumping between multiple separate files.

Call To Action

This document provides an overview of structured learning and certification preparation approaches. For learners seeking clear knowledge organization, guided study planning, and exam-focused practice resources, AAAdemy offers a comprehensive platform to support independent and effective learning.

Explore additional training materials, study guidance, and practice resources at:

https://www.aaademy.com/

Attachment: Answers by Knowledge Point

Develop containerized solutions on Azure

Q1. Correct answer: A
Explanation: The failure occurs at container process startup, so the first evidence should come from the image runtime contract: command, port, and environment. Increasing replicas repeats the same crash, moving the registry does not prove runtime behavior, and creating a search index targets an unrelated dependency.

Q2. Correct answer: C
Explanation: A tag can move, while a digest identifies immutable image content. The environment name and repository description do not identify the running bits, and the base image family cannot prove whether the application layer is current.

Q3. Correct answer: B
Explanation: Container Apps can keep multiple revisions active, and traffic weights decide which revision receives requests. Model parameters, data partitioning, and queue locks do not explain why HTTP users continue to hit an older revision.

Q4. Correct answer: D
Explanation: Scaling depends on the scaler observing the correct metric and authentication path. Blob public access, semantic search, and Event Grid validation are adjacent Azure concepts but do not prove the Container Apps queue-driven scale signal.

Q5. Correct answer: A
Explanation: Timeouts to private endpoints usually point to reachability: DNS, route, firewall, or private endpoint configuration. Re-tagging the image, increasing token limits, or changing queue retry behavior does not validate the network path.

Q6. Correct answer: B
Explanation: A revision captures the deployed template state, including image, environment variables, ingress, probes, scale settings, and traffic assignment. Registry tasks build images, indexers ingest documents, and availability tests observe behavior but do not own the revision template.

Q7. Correct answer: C
Explanation: Image pull authorization is controlled by registry access for the runtime identity. Search ranking, HTTP timeout, and model deployment naming are useful in other scenarios but do not allow the platform to pull the image.

Q8. Correct answer: D
Explanation: Ingress must target the port where the container process listens. ACR digest, Cosmos DB query routing, and Service Bus dead-lettering are separate systems and do not follow from an ingress port mismatch.

Q9. Correct answer: B
Explanation: Scale-from-zero can introduce startup latency before the first worker processes messages. Vector dimensions, secret versions, and telemetry sampling do not explain the initial processing delay after idle periods.

Q10. Correct answer: C
Explanation: Running the image locally verifies startup behavior before registry or platform deployment variables are introduced. Event Grid, Key Vault RBAC, and CPU allocation do not prove the container image contract itself.

Develop AI solutions using Azure data management services

Q1. Correct answer: D
Explanation: Vector search requires the query vector dimension to match the index vector field. Traffic split, queue locks, and Key Vault firewall rules do not explain vector mismatch in retrieval.

Q2. Correct answer: A
Explanation: If ingestion succeeds but lookup by expected IDs fails, the document key and output mappings are the evidence-bearing objects. Replica count, model temperature, and telemetry correlation are unrelated to index document identity.

Q3. Correct answer: B
Explanation: Grounding depends on retrieval inputs and returned evidence: vector field, filters, top-k, and document IDs. Queue partitioning, image base, and secret expiration do not prove whether the RAG query retrieved the intended content.

Q4. Correct answer: C
Explanation: Cosmos DB efficiency depends heavily on partition key design and point-read access patterns. Registry SKU, Event Grid filters, and semantic ranking do not explain high request charge for conversation state lookups.

Q5. Correct answer: D
Explanation: Secure ingestion from Blob Storage requires private access, identity authorization, object metadata, and network reachability. Token limits, replica counts, and queue dead-letter rules do not validate the document source path.

Q6. Correct answer: B
Explanation: Enriched fields depend on skillset outputs being mapped into matching index fields. Revision suffix, registry admin settings, and Cosmos DB consistency do not explain missing enriched search fields.

Q7. Correct answer: C
Explanation: A filter changes the result set, so the first step is to compare field values and document IDs. Restarting compute, granting registry pull permissions, and moving secrets do not validate retrieval filtering.

Q8. Correct answer: D
Explanation: Without the partition key value, Cosmos DB may fan out across partitions. Indexer scheduling, model throttling, and event schema selection are separate concerns.

Q9. Correct answer: A
Explanation: Text and vector fields serve different retrieval mechanics and are not interchangeable. The other options mix unrelated Azure objects or ignore the fixed-dimension requirement of vector fields.

Q10. Correct answer: C
Explanation: Anonymous access exposes source documents and bypasses intended controlled access. Scale rule authentication, vector math, and telemetry correlation are operationally important but do not describe the immediate storage confidentiality risk.

Connect to and consume Azure services

Q1. Correct answer: D
Explanation: SDK client lifetime and retry/timeout settings control outbound dependency behavior under load. Search key mapping, revision traffic, and blob access may matter elsewhere but do not address per-request client churn.

Q2. Correct answer: C
Explanation: A 404 on model invocation commonly points to endpoint, deployment name, API version, or path mismatch. Queue dead-letter state, Key Vault DNS, and replica count do not prove the model-call address is correct.

Q3. Correct answer: B
Explanation: Duplicate processing often occurs when message locks expire or worker logic is not idempotent. Search ranking, registry tags, and Event Grid filters do not own Service Bus message settlement semantics.

Q4. Correct answer: D
Explanation: Event Grid delivery depends on validation, filters, schema handling, and delivery status. Cosmos DB charge, CPU limit, and model temperature do not explain missing Event Grid deliveries.

Q5. Correct answer: A
Explanation: External dependency calls require timeout boundaries, retry classification, and isolation to prevent user-facing stalls. Vector dimensions, image digests, and secret versions do not explain blocked outbound HTTP behavior.

Q6. Correct answer: B
Explanation: Payload parsing failures are controlled by the message contract and can be confirmed through dead-letter reason and worker logs. Telemetry sampling, ingress port, and search top-k are different failure surfaces.

Q7. Correct answer: C
Explanation: Retry behavior should respect throttling evidence such as Retry-After and response metadata. Docker ports, Event Grid validation, and blob metadata do not govern model-call throttling.

Q8. Correct answer: D
Explanation: Secret retrieval belongs in the identity and Key Vault dependency chain before the external API call. Anonymous storage access, image tagging, and max delivery count do not remove static API keys from configuration.

Q9. Correct answer: A
Explanation: Event body shape depends on the selected schema and handler contract. Partition keys, scale cooldown, and analyzers do not define Event Grid payload shape.

Q10. Correct answer: B
Explanation: Timeouts and bounded retries prevent dependency failures from consuming request time indefinitely. Queue name, semantic title, and retention policy do not control how long SDK calls wait.

Secure, monitor, and troubleshoot Azure solutions

Q1. Correct answer: D
Explanation: A 403 indicates the credential exists but lacks required authorization or scope. Chart styling, image size, and Event Grid retry policy do not prove the runtime principal can access the search data plane.

Q2. Correct answer: A
Explanation: Timeouts suggest reachability before permissions. Private DNS, firewall, and network path should be checked before changing roles or rotating secrets. Search, queue, and traffic labels are unrelated to Key Vault network access.

Q3. Correct answer: B
Explanation: Operation id correlation connects request telemetry with dependency calls, exceptions, and traces. Repository name, throughput alone, and subject suffix do not reconstruct the failing request path.

Q4. Correct answer: C
Explanation: The scenario combines throttling and workload pressure, so evidence should come from Retry-After, retry behavior, telemetry, and queue age. Storage access, image digest, and Event Grid validation do not explain throttled model calls plus backlog.

Q5. Correct answer: D
Explanation: Incident causality requires time-ordered evidence across deployments, configuration changes, logs, metrics, alerts, and dependencies. Synonym maps, token settings, and unverified tags do not establish when or why the failure began.

Q6. Correct answer: A
Explanation: Successful token acquisition does not prove the token is for the correct audience or that the principal has the right data-plane scope. Build style, event batching, and duplicate detection do not explain service authorization failure.

Q7. Correct answer: C
Explanation: The configured URI selects the secret object. If it is wrong or deleted, secret retrieval fails for that URI. Revision routing, vector dimensions, and lock renewal are unrelated behaviors.

Q8. Correct answer: B
Explanation: User-visible failures can occur in dependencies even when request logging is incomplete. Dependency and exception telemetry with correlation are needed; registry webhooks, blob tags, and subject filters do not fill that observability gap.

Q9. Correct answer: D
Explanation: Dependency triage needs target, result code, duration, and correlated operations. Analyzer language, repository description, and queue auth rule name do not identify the slow dependency path.

Q10. Correct answer: A
Explanation: Credential chains can select different principals locally and in production. Documenting selected principal and validation evidence avoids confusing local success with runtime authorization. The other options are partial or unrelated.