Exam Code: AI-901
Exam Name: AI-901: Microsoft Azure AI Fundamentals (beta) Training Course
Certification: Microsoft Azure AI Fundamentals (beta)
Vendor: Microsoft

AI-901 Training Course

AI-901: Microsoft Azure AI Fundamentals (beta) Training Course

Structured Learning & Certification Preparation

Table of Contents

Introduction

This study pack is designed to support preparation for the Microsoft Azure AI Fundamentals (beta) exam through a clear, knowledge-point-driven structure. It brings the exam scope into one place so you can review AI concepts, responsible AI requirements, workload patterns, and Microsoft Foundry implementation tasks in the same order you are expected to master them.

The material is organized around 2 official domains, with each domain preserving its detailed operational knowledge points and pairing them with mapped practice questions. A practical way to use this pack is to 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 Azure AI Fundamentals (beta) focuses on recognizing AI workload patterns, responsible AI controls, model capability choices, and basic implementation paths in Microsoft Foundry. The exam expects candidates to connect scenario requirements with the correct AI capability, deployment boundary, tool choice, and validation 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 training course resource that helps learners connect key terms, practical trade-offs, and exam readiness in a format that is practical for steady exam preparation.

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

- Identify AI concepts and responsibilities
- Implement AI solutions by using Microsoft Foundry

Detailed Knowledge Explanation

Identify AI concepts and responsibilities

Apply Responsible AI Requirements to an Azure AI Scenario

Exam Radar

- **Microscopic Technical Focus:** Fairness, reliability, privacy, inclusiveness, transparency, and accountability evidence in solution review
- **Core Priority:** Responsible AI is the policy lens behind design-choice questions: a technically working solution can still be wrong when it hides limitations, collects unnecessary data, excludes users, or lacks ownership for review.
- **Confusion Alert:** The common trap is to choose model accuracy or a larger deployment as the fix when the scenario asks for fairness, privacy, safety, transparency, or accountability controls.

- **Scenario Logic:** Look for who is affected, what decision or recommendation the AI output influences, and whether the missing control is fairness measurement, privacy minimization, disclosure, reliability review, or accountable ownership.
- **Failure Trigger:** The solution may be technically callable but still unsafe when users overtrust output, sensitive data is retained unnecessarily, or no review owner exists for harmful behavior.
- **Operational Dependency:** The blocking dependency is governance evidence attached to the actual user workflow, not a larger model, broader networking change, or more fluent prompt.
- **Topic-Specific Exam Cue:** Words such as fairness, transparency, privacy, responsible use, review, disclosure, sensitive data, or accountability usually mean the answer must name a responsible AI control.

Atomic Deconstruction - Operational Level

Responsible AI questions begin with the affected user and the decision path. The operational object is the control evidence: what data is collected, what limitation is disclosed, who owns review, and which harm is being reduced. Without that evidence, a technically functional model can still be the wrong exam answer because it leaves the risk unmanaged.

The drill is to translate principles into inspection points. Fairness is checked against impacted groups, privacy against prompt and log fields, transparency against user-visible wording, reliability against fallback and validation behavior, and accountability against an owner. The correct option is the one that attaches the principle to a verifiable control.

For exam transformation, treat every option as a proposed dependency. The correct option is the one that unlocks the blocked workflow and can be verified. Wrong options are often useful in another situation, but they fail here because they tune a later step, address a different modality, or repair a symptom without satisfying the scenario requirement.

Component Specifications

For this table, read each row as a simple exam check: before improving the model, verify which responsible AI control is missing and what evidence would prove it exists.

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Impact assessment | Risk category | Fairness, safety, privacy, transparency | Not documented | Use case and affected user group | Review cannot prove which harm is being mitigated |

| Grounding disclosure | User-facing limitation statement | Visible, specific, scenario-bound | Absent unless designed | Application interface and model behavior | Users overtrust generated output or miss human review requirement |

| Data collection boundary | Input retention and minimization rule | Required fields only | Depends on app design | Privacy/security review | Unnecessary personal data enters prompts or logs |

| Accountability owner | Escalation path | Named role or support queue | Undefined | Operational process | Unsafe output has no remediation owner |

Step-by-Step Execution Path

1. Map the user population, decision impact, and sensitive attributes before selecting technical controls. This establishes whether the first dependency is fairness measurement, privacy minimization, or safety review.
2. Inspect prompt, data, and output surfaces for hidden personal data, exclusion risk, unsupported language, and missing confidence or limitation wording.
3. Document the mitigation that directly matches the risk: privacy requires collection limits, transparency requires disclosure, reliability requires validation and fallback, and accountability requires an owner.
4. Verify the review outcome in the project or governance artifact before treating the solution as exam-ready evidence.

Suggested Lab Validation Ideas:

The following paths and commands are conceptual lab-style examples for practice. Adapt them to the current Microsoft documentation, SDK/API version, subscription permissions, and project environment before using them in a real implementation.

- `Portal path: Microsoft Foundry project > Safety or evaluations view` - portal evidence check for safety or quality review status.
- `Local lab script: python review_ai_risk_register.py --scenario support-chat` - local rehearsal for mapping scenario risk to responsible AI principle.

Technical Chain

A scenario starts with an affected user and an AI output path. The selected responsible AI control must attach to that path: privacy limits prompt data before the request is sent, reliability checks validate output before action, transparency changes what the user sees, and accountability defines who responds when the workflow fails. A model-only answer breaks the chain because it changes generation capacity without satisfying the governance dependency.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

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

| Validate privacy minimization | Review prompt fields and application telemetry schema | Only fields required for the task are sent to the model or stored in logs |

| Validate transparency evidence | Inspect UI copy, response metadata, and user disclosure text | The user can see limitations, human review cues, or generated-content notices where required |
| Validate accountability path | Review project risk register or support workflow | Each high-impact failure mode has a named review owner and escalation path |

Select AI Model Capabilities and Deployment Parameters

Exam Radar

- **Microscopic Technical Focus:** Capability matching, model deployment choice, context limits, temperature behavior, and endpoint consumption readiness
- **Core Priority:** AI-901 expects candidates to match workload needs to model capabilities and deployment settings rather than treating every task as a generic chat call.
- **Confusion Alert:** Distractors often pick a famous model name, a high temperature, or a larger context window without proving that the capability matches text, vision, speech, agent, or extraction requirements.
- **Scenario Logic:** Start by identifying whether the app needs text reasoning, image input, embeddings, speech, image generation, or structured extraction; only then evaluate deployment and parameter choices.
- **Failure Trigger:** A request fails or returns unusable output when the selected deployment cannot consume the input modality or when the client targets the wrong deployment name or endpoint.
- **Operational Dependency:** The first dependency is a deployed model with the required capability and a client configuration that can successfully invoke it.
- **Topic-Specific Exam Cue:** Mentions of uploaded images, context size, temperature, deployment name, endpoint, or unsupported input usually point to model capability and deployment-parameter reasoning.

Atomic Deconstruction - Operational Level

Model-selection questions start with capability fit. A deployment is not just a model name; it is the callable endpoint, supported input types, parameter surface, quota context, and authentication path that the app must use.

The drill is to reject parameter tuning until capability is proven. Temperature, token budget, and prompt wording matter only after the model can accept the required modality and the client can route to the right deployment.

For exam transformation, treat every option as a proposed dependency. The correct option is the one that unlocks the blocked workflow and can be verified. Wrong options are often useful in another situation, but

they fail here because they tune a later step, address a different modality, or repair a symptom without satisfying the scenario requirement.

Component Specifications

In fundamentals terms, this table says: check whether the model can produce the required output before choosing temperature, context settings, or code structure.

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Model deployment | Capability family | Chat, multimodal, embedding, image generation, speech | None until deployed | Foundry model catalog and quota | Client cannot call the required capability |

| Context window | Token budget | Model-specific limit | Model dependent | Prompt and retrieved content length | Requests truncate or fail before required evidence is included |
| Temperature | Sampling variability | 0.0 to model-supported maximum | Often 1.0 or service default | Task determinism requirement | Answers become inconsistent for classification or extraction tasks |

| Endpoint credential | Authentication material | Key or identity-supported token | Unavailable until configured | Client application | 401/403 response or missing deployment target |

Step-by-Step Execution Path

1. Classify the task by input and output: text reasoning, image interpretation, speech interaction, visual generation, or structured extraction. This prevents choosing a model that cannot consume the input.
2. Deploy or select a model with the required capability in Microsoft Foundry, then record deployment name, endpoint, and supported parameters.
3. Set deterministic parameters for classification or extraction and more creative parameters only when the scenario asks for varied generation.
4. Run a minimal SDK or REST invocation to prove the endpoint, deployment name, and authentication path are all aligned.

Suggested Lab Validation Ideas:

The following paths and commands are conceptual lab-style examples for practice. Adapt them to the current Microsoft documentation, SDK/API version, subscription permissions, and project environment before using them in a real implementation.

- `Portal path: Microsoft Foundry > Models + endpoints > Deployment details` - portal verification of deployment name, model capability, and endpoint.
- `Python SDK rehearsal: client.responses.create(model=deployment_name, input='Classify this request')` - local code-pattern validation; adapt to the current Foundry

SDK version.

Technical Chain

The client request names a deployment, sends input shaped for a model family, and includes parameters that influence generation. The service resolves the deployment to a hosted model and rejects or degrades the request when capability, authentication, or parameter assumptions do not match. The exam answer must preserve that dependency chain: capability first, deployment identity second, parameter tuning third.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Validate model capability | Microsoft Foundry portal > deployment details | The deployed model supports the required text, image, speech, agent, or generation input |

| Validate endpoint readiness | Run a minimal SDK or REST test call against the deployment | The response returns expected content without 401, 403, 404, or unsupported-parameter errors |
| Validate parameter fit | Inspect request payload for temperature, max output, and input type | Parameter choices match deterministic or creative task requirements |

Identify Azure AI Workload Patterns

Exam Radar

- **Microscopic Technical Focus:** Scenario-to-workload mapping for generative AI, agents, text analytics, speech, vision, image generation, and information extraction
- **Core Priority:** The fundamentals exam asks candidates to recognize which Azure AI workload fits the user goal before choosing a tool or writing code.
- **Confusion Alert:** Wrong options usually choose an adjacent workload: speech synthesis for speech recognition, image generation for computer vision analysis, or chat completion for structured document extraction.
- **Scenario Logic:** Match the source input and required result: transcript, spoken response, generated image, visual interpretation, text insight, generated answer, or structured field extraction.
- **Failure Trigger:** The app uses an AI service that works on related data but produces the wrong output shape, such as a summary where fields are required or synthesized speech where transcription is needed.
- **Operational Dependency:** The blocking dependency is the correct workload pattern, because later configuration cannot repair a service chosen for the wrong transformation.
- **Topic-Specific Exam Cue:** Verbs such as transcribe, synthesize, detect, generate, summarize, classify, or extract usually reveal the workload family.

Atomic Deconstruction - Operational Level

Workload-recognition questions are output-contract questions. The learner should name the data entering the workflow and the exact result the application needs before looking at services.

The drill is to compare adjacent workloads by transformation direction. Speech-to-text, text-to-speech, image interpretation, image generation, text analysis, and information extraction each produce different evidence, so an answer that works on the same input can still be wrong when it returns the wrong output.

For exam transformation, treat every option as a proposed dependency. The correct option is the one that unlocks the blocked workflow and can be verified. Wrong options are often useful in another situation, but they fail here because they tune a later step, address a different modality, or repair a symptom without satisfying the scenario requirement.

Component Specifications

Use this table as a workload decision aid: first ask what the app must return, then choose the service family that naturally returns that result.

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Text analytics workload | Primary output | Entities, sentiment, key phrases, summaries | No output until input text is processed | Language model or text analysis API | App receives prose when it needed labels or extracted values |

| Speech workload | Direction | Speech-to-text or text-to-speech | Not selected | Audio input/output format | The app transcribes when it needed spoken response, or the reverse |
| Vision workload | Input interpretation | Image or video understanding | Not selected | Visual input and model capability | Image is generated or ignored instead of analyzed |

| Information extraction workload | Structure target | Fields, tables, entities from documents/media | Schema absent | Content Understanding configuration | Output cannot be mapped into business fields |

Step-by-Step Execution Path

1. Identify the source input type: plain text, spoken audio, image, video, document, or form. Input type narrows the service family.
2. Identify the required output: generated answer, action plan, transcript, synthetic speech, visual description, generated image, or structured fields.
3. Match the workload to the output evidence, then reject adjacent services that operate on the same input but produce the wrong result.
4. Validate the choice by describing one observable result the app must produce, such as extracted invoice fields or a spoken answer.

Suggested Lab Validation Ideas:

The following paths and commands are conceptual lab-style examples for practice. Adapt them to the current Microsoft documentation, SDK/API version, subscription permissions, and project environment before using them in a real implementation.

- `Local lab script: python classify_workload.py --input invoice.pdf --goal extract-fields` - local rehearsal for scenario mapping.
- `Portal path: Microsoft Foundry > Tools` - portal verification of available Foundry Tools for speech, vision, or Content Understanding workflows.

Technical Chain

A workload decision begins with data modality and ends with an observable output contract. If a question asks for key phrases, the chain goes through text analysis; if it asks for a spoken answer, it requires speech synthesis or a multimodal path; if it asks for fields from documents, it requires extraction. Adjacent services fail because they transform the data into a different output shape.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Validate input modality | Review scenario input and sample payload | Input is categorized as text, audio, image, video, document, or mixed media |

| Validate output contract | Review required application result | The selected workload produces labels, transcript, speech, generated content, visual interpretation, or extracted fields as required |
| Validate distractor rejection | Compare adjacent workload outputs | Rejected options operate on wrong modality or return wrong output shape |

Practice Questions

1. A human resources team uses a Microsoft Foundry-based assistant to recommend internal learning paths. Employees are not told that the recommendation is AI-generated, and no owner is assigned to review complaints about unfair recommendations. What should be addressed first?
   A. Increase the deployment quota for the assistant model
   B. Add a longer system prompt with a friendlier tone
   C. Add transparent user disclosure and define an accountable review path
   D. Move the project to a different Azure region

2. A support app must answer questions about uploaded equipment photos and text descriptions. Which deployment requirement should be verified before tuning model parameters?
   A. The temperature is set to the lowest possible value

B. The deployed model supports multimodal input with images and text
C. The app uses an embedding model for vector similarity only
D. The response format is changed to plain text

3. A finance app must read uploaded invoices and return vendor name, invoice date, line items, tax, and total as fields that can be stored in a database. Which workload pattern best matches the requirement?
   A. Information extraction with Content Understanding
   B. Text-to-speech synthesis
   C. Image generation from a prompt
   D. Sentiment analysis on customer reviews

4. A customer-facing chatbot gives medical-adjacent wellness suggestions. The app stores complete user prompts, including personal details that are not needed for the response. Which responsible AI concern is the first priority?
   A. Increasing context length so the model has more information
   B. Switching to image generation to make the answers more engaging
   C. Raising temperature to make answers less repetitive
   D. Minimizing collected personal data and reviewing prompt/log retention

5. A development team says a deployed chat model can answer all tasks if the prompt is detailed enough. A new feature must compare an uploaded product image with a typed question. What should the exam candidate challenge first?
   A. Whether the model temperature can be increased
   B. Whether the app uses a longer system message
   C. Whether the selected deployment supports the required input modality
   D. Whether the output should include markdown formatting

6. A retail team wants to create product lifestyle images from short text descriptions for marketing drafts. Which workload should they choose?
   A. Speech recognition
   B. Image generation
   C. Key phrase extraction
   D. Document intelligence extraction

7. A call-center solution receives recorded calls and must produce transcripts for agents to review. Which service direction is required first?
   A. Speech-to-text
   B. Text-to-speech
   C. Image-to-text visual reasoning
   D. Prompt-to-image generation

8. A model returns inconsistent labels for the same short classification prompt. The deployment supports the required text input, and the endpoint call succeeds. Which parameter adjustment is

most appropriate?
A. Increase temperature to encourage more varied answers
B. Replace the classifier with text-to-speech
C. Use an image generation model for the labels
D. Lower randomness and use a more deterministic request configuration

9. A project team wants users with screen readers to understand an AI-generated summary and know when human review is available. Which responsible AI principle is most directly involved?
A. Capacity planning only
B. Image generation quality
C. Inclusiveness and transparency
D. Larger context windows

10. A travel app needs to detect whether reviews mention location, staff, or cleanliness and return those phrases for downstream reporting. Which workload pattern is the best fit?
A. Text-to-speech
B. Key phrase or entity extraction from text
C. Image generation
D. Speech synthesis before analysis

Implement AI solutions by using Microsoft Foundry

Build and Test a Foundry Chat and Single-Agent Solution

Exam Radar

- **Microscopic Technical Focus:** System prompt design, model deployment interaction, lightweight chat client, agent instruction, tool boundary, and test trace validation
- **Core Priority:** The largest AI-901 domain includes practical Foundry tasks: deploying a model, interacting with it, creating a lightweight client, and testing a single-agent flow.
- **Confusion Alert:** Distractors often skip the deployment name, confuse prompt engineering with agent tool wiring, or change code before validating behavior in the Foundry portal.
- **Scenario Logic:** Separate prompt behavior from runtime routing: the prompt or agent instruction controls behavior, while endpoint, deployment name, agent ID, credential, and tool configuration control whether the app reaches the correct runtime object.
- **Failure Trigger:** The app returns routing errors, ignores tool boundaries, or produces off-scope answers when the deployment identity, agent instruction, or tool configuration is not validated in the portal first.
- **Operational Dependency:** The first dependency is a verified Foundry deployment or agent test, followed by client configuration that exactly matches the tested object.

- **Topic-Specific Exam Cue:** Questions that mention playground success, deployment-not-found errors, agent tests, tools, system prompts, or lightweight clients usually ask for the first validation point.

Atomic Deconstruction - Operational Level

Foundry chat and agent questions are runtime-routing questions. The core objects are the system prompt or agent instruction, the deployment or agent ID, configured tools, and the client settings that send the request to the intended runtime object.

The drill is to validate in the portal before expanding code. If the playground or agent test works, a client failure usually points to endpoint, deployment name, agent ID, or credential mismatch. If the portal behavior is wrong, the instruction or tool boundary must be corrected first.

For exam transformation, treat every option as a proposed dependency. The correct option is the one that unlocks the blocked workflow and can be verified. Wrong options are often useful in another situation, but they fail here because they tune a later step, address a different modality, or repair a symptom without satisfying the scenario requirement.

Component Specifications

For a beginner, this table means: prove the Foundry object works in the portal, then make sure the client is calling that exact object.

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| System prompt | Behavior constraint | Role, boundaries, response format, safety rule | Empty or generic | Model request | Model produces off-scope or unstructured responses |

| Model deployment | Invocation target | Deployment name and endpoint | Not callable until created | Client SDK configuration | 404 deployment not found or wrong model family |
| Agent instruction | Task policy | Goal, allowed actions, tool limits | Undefined | Agent runtime | Agent chooses unsupported actions or ignores constraints |

| Client application | Configuration source | Endpoint, deployment/agent ID, credential | Local placeholder | Environment variables and SDK | Authentication or routing failure |

Step-by-Step Execution Path

1. Create the system prompt or agent instruction before testing so the model has a stable behavioral contract.
2. Deploy or select the model in Microsoft Foundry and verify the deployment name from the portal instead of inventing a model identifier.

3. Test the prompt or single-agent solution in the Foundry portal to observe response quality, tool selection, and error messages before coding.
4. Build the lightweight client with endpoint, credential, and deployment or agent identifier stored in configuration.
5. Run a minimal conversation and inspect response text, status, and trace or portal test output to confirm the app calls the intended Foundry object.

Suggested Lab Validation Ideas:

The following paths and commands are conceptual lab-style examples for practice. Adapt them to the current Microsoft documentation, SDK/API version, subscription permissions, and project environment before using them in a real implementation.

- `Portal path: Microsoft Foundry > Models + endpoints > Playground` - portal test for model deployment response behavior.
- `Portal path: Microsoft Foundry > Agents > Test` - portal test for single-agent instruction and tool behavior.
- `Python SDK rehearsal: python app.py` - local client validation after setting endpoint, credential, and deployment or agent ID environment variables.

Technical Chain

The system prompt or agent instruction sets the behavior contract. The client request then targets a Foundry deployment or agent endpoint with credentials. The runtime loads the model and any configured agent tools, produces a response, and records evidence in the portal test surface or trace. If the deployment name, agent ID, or instruction is wrong, the failure appears as routing errors, unsupported behavior, or off-scope output.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| --- | --- | --- |
| Validate deployment identity | Microsoft Foundry portal > Models + endpoints > deployment details | Deployment name in code exactly matches the Foundry deployment |

| Validate agent behavior | Microsoft Foundry portal > Agents > Test conversation | Agent follows instruction boundaries and uses only configured tools |
| Validate client call path | Run lightweight client and inspect response/status | Client receives a successful response from the intended deployment or agent |

Implement Text and Speech Solutions with Foundry Tools

Exam Radar

- **Microscopic Technical Focus:** Text analysis output contracts, spoken prompt response, Azure Speech tool selection, audio format, and lightweight app validation
- **Core Priority:** AI-901 includes practical text and speech implementation, so candidates must distinguish text analysis, speech recognition, speech synthesis, and multimodal spoken interactions.
- **Confusion Alert:** A common wrong answer is to use a generative chat model for every language task or to choose speech synthesis when the requirement is transcribing spoken input.
- **Scenario Logic:** Determine the direction of the language or audio transformation: text to labels, audio to text, text to audio, or a combined spoken interaction.
- **Failure Trigger:** The workflow breaks when raw audio is sent to a text-only path, when synthesis is chosen before transcription, or when the response format does not match the application need.
- **Operational Dependency:** The blocking dependency is the correct direction and output contract for Speech or text analysis before voice, language, or formatting options are tuned.
- **Topic-Specific Exam Cue:** Words such as listen, transcribe, speak, voice, sentiment, entities, key phrases, or spoken answer usually identify the required tool direction.

Atomic Deconstruction - Operational Level

Text and speech implementation questions are direction-sensitive. Spoken input must become text before a text-only reasoning step can use it, while a spoken answer requires generated or selected text to become audio.

The drill is to label the conversion direction and the expected response field. A transcript, sentiment label, key phrase list, synthesized audio stream, and spoken answer are different completion states, so the selected tool must create the one requested by the scenario.

For exam transformation, treat every option as a proposed dependency. The correct option is the one that unlocks the blocked workflow and can be verified. Wrong options are often useful in another situation, but they fail here because they tune a later step, address a different modality, or repair a symptom without satisfying the scenario requirement.

Component Specifications

Read this table as a direction check: decide whether the app is converting speech to text, text to speech, or text to analysis output before writing code.

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

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

| Text analysis call | Output type | Key phrases, entities, sentiment, summary | No analysis until requested |
Text input and service capability | Application receives free-form text instead of required labels |

| Speech recognition | Direction | Audio to text | Not configured | Audio stream and language | Spoken input is not converted into prompt text |
| Speech synthesis | Direction | Text to audio | Not configured | Voice selection and output format | Application cannot return spoken responses |

| Audio format | Encoding/sample assumptions | Service-supported WAV/PCM or SDK stream | Client dependent | Speech call fails or returns poor recognition | undefined |

Step-by-Step Execution Path

1. Write the required output contract first: sentiment label, extracted entities, recognized transcript, or spoken answer.
2. Select the Foundry Tool or multimodal model path that matches the direction of the audio or text transformation.
3. Configure language, voice, and audio format only after the workload direction is correct.
4. Run a minimal app invocation and verify the returned transcript, text labels, or playable speech output before expanding the workflow.

Suggested Lab Validation Ideas:

The following paths and commands are conceptual lab-style examples for practice. Adapt them to the current Microsoft documentation, SDK/API version, subscription permissions, and project environment before using them in a real implementation.

- `Portal path: Microsoft Foundry > Tools > Speech` - portal verification of speech capability and configuration.
- `Python SDK rehearsal: python speech_demo.py --input prompt.wav` - local validation for audio-to-text or spoken-response flow; adapt to current SDK package names.
- `Python SDK rehearsal: python text_analysis_demo.py --text reviews.txt` - local validation for key phrase, entity, sentiment, or summary output.

Technical Chain

Audio and text solutions are directional. A microphone input must become text before a text-only model can reason over it, while a spoken answer requires generated text to be synthesized into audio. Text analysis tasks are evaluated by structured labels or extracted values. The exam answer must keep input direction, service capability, and expected output connected.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Validate speech direction | Inspect app requirement and Speech tool configuration | Configuration is speech-to-text for prompts or text-to-speech for spoken output |

| Validate text analysis contract | Run sample input through the text analysis call | Response contains the required entities, key phrases, sentiment, or summary fields |
| Validate audio compatibility | Inspect audio format or SDK stream settings | Audio input is accepted and produces transcript or playable output |

Implement Vision and Image Generation Solutions with Foundry

Exam Radar

- **Microscopic Technical Focus:** Visual prompt interpretation, multimodal model input, generated image output, vision app capability, and result validation
- **Core Priority:** The exam expects candidates to separate vision understanding from image generation and to know when a deployed multimodal model is required.
- **Confusion Alert:** Distractors choose image generation when the app must understand an image, or choose OCR-only extraction when the scenario asks for broad visual interpretation.
- **Scenario Logic:** Decide whether the workflow inspects an existing image or creates a new image from a prompt, then validate that the selected model accepts or returns the correct visual artifact.
- **Failure Trigger:** The app fails conceptually when image generation is selected for image understanding, or when a text-only model is asked to reason over visual input.
- **Operational Dependency:** The first dependency is the transformation direction: image-to-answer for vision understanding, or prompt-to-image for generation.
- **Topic-Specific Exam Cue:** Interpret, describe, inspect, detect, and identify point to vision understanding; generate, create, or produce an image points to image generation.

Atomic Deconstruction - Operational Level

Vision and image generation questions are modality-transformation questions. Vision interpretation starts from an existing visual input and returns labels or reasoning; image generation starts from a prompt and returns a new visual artifact.

The drill is to read scenario verbs carefully. Interpret, describe, identify, and detect point to visual understanding. Generate, create, or produce a picture points to image generation. The validation evidence must match that direction.

For exam transformation, treat every option as a proposed dependency. The correct option is the one that unlocks the blocked workflow and can be verified. Wrong options are often useful in another situation, but they fail here because they tune a later step, address a different modality, or repair a symptom without satisfying the scenario requirement.

Component Specifications

For exam use, this table helps separate two easy-to-confuse choices: understanding an image that already exists versus generating a new image.

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Visual input | Prompt attachment | Image plus text instruction | Not included unless sent | Multimodal deployment | Model answers without seeing image evidence |

| Vision interpretation | Result type | Description, classification, detected content | Depends on prompt and model | Image quality and capability | Wrong labels or missing visual reasoning |
| Image generation | Output artifact | Generated image from prompt | No image until model call | Generative image model and safety filters | No visual asset is produced |

| Safety filter | Content policy response | Allowed, revised, or blocked | Service controlled | Prompt and policy | Request is blocked or altered without handling |

Step-by-Step Execution Path

1. Decide whether the app must interpret an existing image or create a new image. This is the critical workload split.
2. For interpretation, send the visual input with a text instruction to a deployed multimodal model and verify that the response references visible evidence.
3. For generation, send a precise prompt to an image generation model and handle safety or policy responses.
4. Record the output contract: classification, description, detected attribute, or generated asset location.

Suggested Lab Validation Ideas:

The following paths and commands are conceptual lab-style examples for practice. Adapt them to the current Microsoft documentation, SDK/API version, subscription permissions, and project environment before using them in a real implementation.

- `Portal path: Microsoft Foundry > Playground > multimodal test` - portal validation that image plus prompt input is accepted.
- `Python SDK rehearsal: python vision_prompt.py --image meter.jpg --question 'What reading is visible?'` - local validation for visual interpretation.
- `Python SDK rehearsal: python image_generate.py --prompt 'product mockup on white background'` - local validation for generated image output.

Technical Chain

Vision interpretation sends existing pixels to a model that can inspect visual features and combine them with the text instruction. Image generation starts with text and returns new pixels. The underlying request shape is therefore different. A correct exam choice follows the direction of transformation: image-to-answer for vision, prompt-to-image for generation.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Validate multimodal input | Run a visual prompt test with an attached image | Response uses visible image evidence rather than generic text |

| Validate generated asset | Inspect image generation response or output file path | A new image artifact is returned or a policy response is handled |
| Validate workload split | Compare scenario verb: interpret, describe, detect, generate, create | Selected service direction matches the requested transformation |

Extract Information with Azure Content Understanding in Foundry Tools

Exam Radar

- **Microscopic Technical Focus:** Schema-driven extraction from documents, images, audio, and video using Content Understanding and lightweight client validation
- **Core Priority:** AI-901 specifically includes Content Understanding for information extraction across documents, forms, images, audio, and video.
- **Confusion Alert:** Wrong options often use generic chat summarization or vision description when the requirement is a repeatable schema with fields that an app can consume.
- **Scenario Logic:** Look for repeatable business fields from documents, images, audio, or video rather than a general natural-language answer.
- **Failure Trigger:** Generic chat or summarization returns prose that cannot reliably populate application fields, preserve confidence evidence, or trigger human review.
- **Operational Dependency:** The first dependency is a schema or extraction objective that returns structured fields the app can map and validate.
- **Topic-Specific Exam Cue:** Field names, tables, invoice data, call attributes, dates, amounts, confidence, analyzer, or structured output usually signal Content Understanding.

Atomic Deconstruction - Operational Level

Content Understanding questions are schema-evidence questions. The operational object is not a paragraph summary; it is the repeatable extraction result that an application can map into fields, tables, confidence checks, and review workflows.

The drill is to design from the target fields backward. If the app needs customer name, issue category, line items, dates, or amounts, the correct answer must preserve schema, status, confidence, and client mapping rather than only producing natural-language output.

For exam transformation, treat every option as a proposed dependency. The correct option is the one that unlocks the blocked workflow and can be verified. Wrong options are often useful in another situation, but they fail here because they tune a later step, address a different modality, or repair a symptom without satisfying the scenario requirement.

Component Specifications

For this table, think like an app developer: the key question is whether the tool returns fields your program can store, check, and send for review.

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Extraction schema | Target fields | Names, types, tables, confidence | Absent until defined | Business output contract | Response is unstructured and cannot populate the app |

| Content source | Media type | Document, image, audio, video | Not uploaded or referenced | Tool support and file quality | No extraction or partial extraction |
| Analyzer/test run | Validation state | Succeeded, failed, needs review | Untested | Sample content and schema | Fields are missing before client integration |

| Client mapping | Field binding | JSON field to app property | Manual or absent | Extraction response | Application stores wrong values or drops confidence signals |

Step-by-Step Execution Path

1. Define the business fields that must be extracted before choosing the tool. This turns the requirement into a schema rather than a broad summary.
2. Use Content Understanding in Foundry Tools with representative documents, images, audio, or video to test extraction quality.
3. Inspect field values, confidence, and missing-field behavior so the app can decide when human review is required.
4. Build a lightweight client that submits content, reads structured output, maps fields to application objects, and logs extraction failures.

Suggested Lab Validation Ideas:

The following paths and commands are conceptual lab-style examples for practice. Adapt them to the current Microsoft documentation, SDK/API version, subscription permissions, and project environment before using them in a real implementation.

- `Portal path: Microsoft Foundry > Tools > Content Understanding` - portal validation for analyzer/schema and test results.
- `Python SDK rehearsal: python extract_content.py --file invoice.pdf` - local client validation for structured extraction; adapt to current Content Understanding SDK/API surface.
- `API rehearsal: inspect JSON response fields, confidence scores, and status` - response-shape validation rather than deployment command.

Technical Chain

The app sends a content item to an analyzer that applies a schema or extraction objective. The service processes the media, returns structured fields with status and confidence evidence, and the client maps those fields into business objects. If the answer uses generic summarization, the chain loses schema, confidence, and repeatable field binding.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Validate schema coverage | Microsoft Foundry > Tools > Content Understanding analyzer view | All required business fields are represented in the analyzer or extraction output |

| Validate extraction result | Run a sample file and inspect JSON/status output | Expected fields, values, and confidence indicators are present |
| Validate client mapping | Review application mapping from extraction response to data model | Each required field is stored correctly and low-confidence cases are handled |

Practice Questions

1. A developer successfully tests a chat model in the Microsoft Foundry playground, but the application receives a deployment-not-found error. What should the developer verify first?
   A. Whether the prompt contains enough examples
   B. Whether the temperature value is above 1.0
   C. Whether the app should use image generation instead
   D. Whether the endpoint, deployment name, and API version match the tested deployment

2. A lobby kiosk must listen to a visitor question, find an answer, and speak the answer aloud. Which sequence best matches the requirement?

A. Recognize speech to text, generate or retrieve the answer, then synthesize the answer to speech
B. Synthesize speech first, then transcribe the generated audio
C. Send raw microphone audio to a text-only chat deployment without conversion
D. Generate an image and describe it to the visitor

3. An inspection app must analyze a photo of a control panel and answer whether a warning indicator is visible. What capability is required?
A. Text-to-speech synthesis
B. A deployed multimodal model that can receive the image and instruction
C. A prompt-to-image generation model
D. A document table extraction schema only

4. A lightweight claims app submits repair photos, PDFs, and short videos and must return claim number, damage category, date, amount, and confidence scores. What should the developer validate first in Foundry Tools?
A. That a generic chat prompt produces a paragraph summary
B. That text-to-speech can read the claim aloud
C. That Content Understanding returns the required structured fields from representative files
D. That an image generation model can create a replacement asset

5. A single-agent prototype in Microsoft Foundry answers correctly in the playground. Before integrating it into a web app, which evidence most directly proves the client can call it?
A. A screenshot of the prompt wording
B. A list of possible future agent features
C. A larger context-window model selection
D. A minimal SDK or REST invocation that succeeds against the configured endpoint and deployment

6. A training app takes written learner feedback and must return sentiment labels plus key phrases for each comment. Which implementation choice aligns with the required output?
A. Use a text analysis path that returns sentiment and key phrase fields
B. Use image generation to visualize each comment
C. Use speech synthesis as the primary analysis step
D. Use a vision model because text comments are user-generated

7. A designer asks an app to create a new banner image from the prompt "modern dashboard on a clean desk." Which validation confirms the right Foundry capability is being used?
A. The response includes extracted invoice totals
B. The image generation response returns a new image artifact or a handled policy response
C. The speech recognizer returns a transcript
D. The chat response refuses to accept image input

8. A Content Understanding analyzer returns values for most invoice fields, but confidence is low for total amount on scanned copies. What should the application logic do first?
   A. Ignore confidence and store every value automatically
   B. Switch to text-to-speech for scanned invoices
   C. Route low-confidence extraction results for review or validation
   D. Increase chat temperature to make the amount more creative

9. A developer sends a text-only request to a deployed model and receives an unsupported-parameter error. Which check should happen before rewriting the whole application?
   A. Whether the selected endpoint supports the request payload and parameters being sent
   B. Whether the app should create images instead of text
   C. Whether the project needs more marketing copy
   D. Whether speech synthesis has a different voice

10. A media review app must process short videos and return scene category, visible brand, spoken product name, and confidence values as fields. Which Foundry approach is the best first fit?
    A. Text-to-speech using a natural voice
    B. A chat-only paragraph summary with no field mapping
    C. Image generation from a descriptive prompt
    D. Content Understanding with a schema or extraction objective for the required fields

---

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.
- Study the responsible AI and workload-pattern sections before implementation sections so scenario requirements are clear before tool selection.
- In each technical section, follow the Exam Radar, Component Specifications, Step-by-Step Execution Path, Technical Chain, and Operational Skills Matrix in order.
- Answer the Practice Questions immediately after each domain and review the answer attachment only after making your own selection.
- Revisit weak areas by mapping every wrong answer back to the missed dependency: responsible AI evidence, model capability, input modality, endpoint configuration, or structured output requirement.

Who This PDF Is For

This study pack is intended for learners preparing for the Microsoft Azure AI Fundamentals (beta) exam who want a structured, exam-aligned review resource. It is especially useful for cloud developers, AI beginners,

solution architects, students, and Azure practitioners who need to connect Azure AI concepts with Microsoft Foundry implementation choices.

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.

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

Identify AI concepts and responsibilities

Q1. Correct answer: C
Explanation: C is correct because the scenario shows transparency and accountability gaps in a user-impacting workflow. A may improve capacity but not disclosure or ownership. B may improve style but not governance evidence. D changes placement and may matter for compliance, but it does not solve the missing review responsibility.

Q2. Correct answer: B
Explanation: B is correct because the app needs a model that can actually receive and reason over image plus text input. A tunes randomness after capability is proven. C supports similarity search, not direct visual interpretation by itself. D changes output formatting but cannot make a text-only model process an image.

Q3. Correct answer: A
Explanation: A is correct because the requirement is repeatable structured field extraction from documents. B creates audio from text. C creates new images rather than reading invoice fields. D classifies opinion or tone and does not populate invoice data.

Q4. Correct answer: D
Explanation: D is correct because the issue is unnecessary sensitive data entering prompts or logs. A may increase exposure rather than reduce it. B changes workload type and misses the privacy risk. C changes response variability but does not address data minimization.

Q5. Correct answer: C
Explanation: C is correct because capability fit comes before prompt design; a text-only deployment cannot inspect an uploaded image. A affects randomness after the request is valid. B cannot add unsupported image input capability. D changes presentation and does not solve modality mismatch.

Q6. Correct answer: B
Explanation: B is correct because the task starts with text and requires a new visual asset. A converts audio to text. C extracts terms from text but does not create images. D extracts fields from documents and does not generate marketing visuals.

Q7. Correct answer: A
Explanation: A is correct because recorded spoken audio must be converted into text transcripts. B converts text into spoken audio and is the opposite direction. C analyzes images, not calls. D creates images and does not process speech.

Q8. Correct answer: D
Explanation: D is correct because classification usually needs stable output after the correct text-capable deployment is already working. A makes variability worse. B changes the workload to audio output. C targets visual generation and cannot solve text label consistency.

Q9. Correct answer: C
Explanation: C is correct because the scenario combines accessible user experience with clear disclosure and review cues. A concerns throughput, not user access or disclosure. B is unrelated to a text summary interface. D may help long prompts but does not address accessibility or transparency.

Q10. Correct answer: B
Explanation: B is correct because the app needs structured insight from text. A and D create spoken audio and do not extract phrases from reviews. C creates images and is unrelated to review analysis.

Implement AI solutions by using Microsoft Foundry

Q1. Correct answer: D
Explanation: D is correct because the portal test proves the deployment can work, so the client route and deployment identity are the first dependencies to validate. A and B affect generation behavior after routing succeeds. C changes workload type without addressing the 404-style deployment mismatch.

Q2. Correct answer: A
Explanation: A is correct because the workflow has audio input, text reasoning or retrieval, and audio output. B reverses the dependency. C fails when the model path cannot consume raw audio. D uses the wrong modality and output contract.

Q3. Correct answer: B
Explanation: B is correct because the app must interpret an existing image and answer from visible evidence. A converts text to audio. C creates new images rather than inspecting uploaded photos. D is useful for structured document extraction, not broad visual reasoning over equipment images.

Q4. Correct answer: C
Explanation: C is correct because the application needs repeatable structured extraction with confidence

evidence across content types. A may summarize but does not guarantee field-level mapping. B changes text to audio and misses extraction. D creates images and does not populate claim fields.

Q5. Correct answer: D
Explanation: D is correct because integration depends on a callable endpoint, deployment name, authentication path, and API version. A documents prompt text but not client readiness. B is planning information, not validation. C may help long inputs but does not prove the app can invoke the deployment.

Q6. Correct answer: A
Explanation: A is correct because the expected output is text-analysis labels and phrases. B creates images and does not analyze the text. C creates audio and is not the analysis step. D targets visual input and is unrelated to written comments.

Q7. Correct answer: B
Explanation: B is correct because prompt-to-image generation should return a generated visual artifact or a policy-handling result. A validates document extraction, not generation. C validates speech-to-text. D describes a failed modality path, not successful image generation.

Q8. Correct answer: C
Explanation: C is correct because confidence evidence must drive review or validation when extracted fields may be unreliable. A drops the operational safeguard. B changes the modality output and does not improve extraction confidence. D makes generation more variable and is inappropriate for numeric fields.

Q9. Correct answer: A
Explanation: A is correct because unsupported-parameter errors often come from a mismatch between deployment capability, API version, and request payload. B changes the workload. C is unrelated to runtime failure. D affects audio output and does not explain a text request parameter error.

Q10. Correct answer: D
Explanation: D is correct because the app needs structured extraction from video content with field and confidence evidence. A produces audio from text. B may produce prose but loses schema, confidence, and repeatable client mapping. C creates new images rather than analyzing video.