Exam Code: AI-300
Exam Name: AI-300: Microsoft Certified Machine Learning Operations MLOps Engineer Associate Training Course
Certification: Microsoft Certified Machine Learning Operations MLOps Engineer Associate
Vendor: Microsoft

AI-300 Training Course

AI-300: Microsoft Certified Machine Learning Operations MLOps Engineer Associate Training Course

Structured Learning & Certification Preparation

Table of Contents

Introduction

This study pack is designed to support preparation for the Microsoft Certified Machine Learning Operations MLOps Engineer Associate exam through a clear, knowledge-point-driven structure. It brings the exam scope into one place so you can review Design and implement an MLOps infrastructure, Implement machine learning model lifecycle and operations, Design and implement a GenAIOps infrastructure, Implement generative AI quality assurance and observability, and related domains in the same order you are expected to master them.

The material is organized around 5 official knowledge points, 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 Machine Learning Operations MLOps Engineer Associate focuses on the ability to understand the core concepts, terminology, roles, operational practices, and decision-making patterns covered by the certification blueprint. The exam expects candidates to connect foundational knowledge with practical scenarios and choose actions that fit the stated business, technical, and operational context.

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 key terms, domain concepts, practical trade-offs, and exam readiness in a format that is practical for steady exam 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

- Design and implement an MLOps infrastructure
    - Create and manage resources in an Azure Machine Learning workspace
    - Create and manage assets in an Azure Machine Learning workspace
    - Implement infrastructure as code for Azure Machine Learning
- Implement machine learning model lifecycle and operations
    - Orchestrate model training in Azure Machine Learning
    - Implement model registration and versioning
    - Deploy machine learning models for production environments
    - Monitor and maintain machine learning models in production
- Design and implement a GenAIOps infrastructure
    - Implement Foundry environments and platform configuration

    - Deploy and manage foundation models for production workloads
    - Implement prompt versioning and management with source control
- Implement generative AI quality assurance and observability
    - Configure evaluation and validation for generative AI applications and agents
    - Implement observability for generative AI applications and agents
- Optimize generative AI systems and model performance
    - Optimize retrieval-augmented generation performance and accuracy
    - Implement advanced fine-tuning and model customization

Detailed Knowledge Explanation

Design and implement an MLOps infrastructure

Create and manage resources in an Azure Machine Learning workspace

Exam Radar

- Official Blueprint Mapping: Create and manage a workspace; Create and manage datastores; Create and manage compute targets; Configure identity and access management for workspaces.
- Domain Weight: Design and implement an MLOps infrastructure represents 15-20% of the official skills measured.
- Core Priority: The exam tests whether the candidate can operate this sub-skill as a concrete Azure workflow inside Design and implement an MLOps infrastructure, not merely identify the service name.
- High Frequency: Expect scenario wording that combines Machine Learning workspace, Workspace managed identity, Datastore, and verification evidence from commands, logs, metrics, or portal state.
- Confusion Alert: A contributor can create workspace objects but still fail data reads when the job identity lacks storage data-plane access.
- Scenario Logic: Choose the answer that creates a verifiable dependency chain: configure the object, bind identity or version, run the operation, then prove the resulting state.
- Version Delta: AI-300 combines Azure Machine Learning and Microsoft Foundry under AIOps, so this point must be read as an operational platform task rather than a standalone concept.
- Failure Trigger: Training jobs queue or fail when compute quota, datastore identity, private DNS, or storage firewall dependencies are missing.

- Operational Dependency: The task depends on Machine Learning workspace, Workspace managed identity, Datastore, Compute cluster, Storage account, Private endpoint and a validation step that proves the configured state is actually usable.
- How the Exam Asks It: A company creates an Azure Machine Learning workspace, but training jobs cannot mount data from a locked-down storage account. The question asks which workspace dependency or identity permission must be configured before jobs can run.
- How Distractors Are Designed: Distractors point to workspace Contributor permissions, increasing compute size, enabling Application Insights, or changing the experiment name. These are plausible Azure ML actions, but none grants the job identity data-plane access through the storage boundary.
- Why the Correct Answer Works: The correct answer binds the workspace or compute managed identity to the storage account with the required Storage Blob Data role and validates datastore access, because the runtime needs data-plane authorization after the workspace exists.

Atomic Deconstruction - Operational Level

Microscopic technical focus: Provisioning workspace dependencies, datastore bindings, compute targets, and IAM boundaries.

Beginner explanation: Think of the workspace as the operations control room. It records jobs and assets, but the actual data, secrets, images, and compute capacity live in dependent Azure resources that must be permissioned separately.

Operational split for this point: start with Machine Learning workspace, then verify Workspace managed identity and Datastore before trusting any production outcome. The exam is testing whether the candidate can locate the missing dependency, not whether the candidate recognizes every service name in the scenario.

For this knowledge point, the target objects are Machine Learning workspace, Workspace managed identity, Datastore, Compute cluster, Storage account, Private endpoint. The exam usually describes one broken link in that chain. The correct answer is the option that restores the missing operational dependency rather than the option that only describes the platform at a high level.

Why-layer: Machine Learning workspace becomes exam-relevant only when the surrounding dependency chain can run. In this topic, Training jobs queue or fail when compute quota, datastore identity, private DNS, or storage firewall dependencies are missing. The correct configuration matters because it changes the state that controls execution, authorization, resolution, evaluation, or observability; a nearby but unrelated action leaves the same failure mode in place.

Decision tree: if the scenario describes access failure, inspect identity and RBAC before changing compute or code; if it describes unresolved assets, inspect name, version, and scope; if it describes runtime failure, inspect logs, endpoint invocation, metrics, or evaluation output; if it describes quality degradation, inspect data, retrieval, evaluation, and monitoring evidence before changing the model.

Common mistakes: Confusing workspace Contributor with Storage Blob Data Contributor. Creating a datastore before verifying that the job identity can read the storage container. Forgetting private DNS when

public access is disabled.

Practice question: A company creates an Azure Machine Learning workspace, but training jobs cannot mount data from a locked-down storage account. The question asks which workspace dependency or identity permission must be configured before jobs can run.
A. Assign Storage Blob Data Contributor to the workspace or compute managed identity on the storage scope, then validate the datastore with a smoke-test job.
B. Assign Contributor on the Azure Machine Learning workspace to the data scientist group.
C. Increase the compute cluster maximum node count to provide more training capacity.
D. Enable Application Insights on the workspace and review request telemetry.

expected answer: A

Explanation: A is correct because the failure is data-plane storage authorization during job execution. B affects management-plane workspace operations, C affects capacity, and D improves observability but does not grant blob access.

The common decision point is: A contributor can create workspace objects but still fail data reads when the job identity lacks storage data-plane access. Therefore, read every scenario for the actor, the resource scope, the object version, the network path, the metric threshold, and the expected observable result.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Machine Learning workspace | Managed identity | System-assigned or user-assigned | Identity absent until enabled or assigned | Entra ID principal creation | Jobs cannot authenticate to protected datastores |

| Datastore | Credential mode | Account key, SAS, service principal, user identity, managed identity | Metadata only until credential or identity path works | Storage account, container, RBAC, firewall | Mount or download step fails with authorization error |
| Compute cluster | Node range | min=0 or greater, max within quota | Scaled down when idle | Regional VM quota and workspace | Job remains queued or provisioning fails |

| Private endpoint | DNS resolution | Approved private IP with linked private DNS zone | Public endpoint used unless restricted | VNet, subnet, DNS zone, firewall | Studio or job traffic cannot reach workspace or storage |

Step-by-Step Execution Path

1. Execute the operational step.
   az ml workspace show -g rg-ai300 -n mlw-ai300-dev --query identity.principalId -o tsv
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Read the workspace principal ID first because role assignment cannot target an identity that has not materialized in Entra ID.

Checkpoint: A non-empty principal ID is returned.

2. Execute the operational step.
   az role assignment create --assignee --role "Storage Blob Data Contributor" --scope
   Command type: Azure CLI RBAC verification for Entra identity and Azure AI resource scope.

Reason: Assign data-plane storage access because workspace Contributor does not authorize blob reads inside the training container.

Checkpoint: Role assignment list shows Storage Blob Data Contributor at the storage account or container scope.

3. Execute the operational step.
   az ml datastore create --file datastore.yml -g rg-ai300 -w mlw-ai300-dev
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Register the datastore after RBAC is available so the datastore reference resolves to a storage path the job identity can actually use.

Checkpoint: Datastore show output contains the expected account, container, and identity-based access mode.

4. Execute the operational step.
   az ml job create --file train-smoke-test.yml -g rg-ai300 -w mlw-ai300-dev
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Submit a small smoke-test job because datastore creation alone does not prove runtime mount behavior.

Checkpoint: The job reaches Completed and logs show successful data access.

Technical Chain

A user, workflow, or deployment command targets Machine Learning workspace and submits configuration to Azure control plane or a project runtime. Azure validates identity, resource scope, quota, version references, and network reachability because the runtime cannot safely use an object that is not authorized, versioned, reachable, or measurable. The configured object then participates in the runtime path through Workspace managed identity, Datastore, Compute cluster. This sequence works because each object unlocks the next dependency: identity allows access, versioning allows reproducibility, network resolution allows execution, and telemetry allows verification. When the workload executes, telemetry, status output, logs, API response, or evaluation metrics prove whether the chain is complete. If the chain breaks, the failure appears as the operational symptom described in the scenario: Training jobs queue or fail when compute quota, datastore identity, private DNS, or storage firewall dependencies are missing. An incorrect configuration creates the observed failure because it changes a nearby object while leaving the actual missing dependency unresolved.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

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

| Confirm workspace identity | `az ml workspace show -g rg-ai300 -n mlw-ai300-dev --query identity.principalId -o tsv` | A principal ID is returned. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

| Verify storage data-plane role | `az role assignment list --assignee <principal-id> --scope <storage-id> -o table` | Storage Blob Data Contributor appears at the expected scope. Command type: Azure CLI RBAC verification for Entra identity and Azure AI resource scope. |

| Inspect datastore binding | `az ml datastore show -n trainingdata -g rg-ai300 -w mlw-ai300-dev` | Account, container, and credential mode match the intended design. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

| Prove runtime data access | `az ml job show --name <smoke-test-job> -g rg-ai300 -w mlw-ai300-dev --query status` | Smoke-test job reaches Completed after reading the datastore. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

Create and manage assets in an Azure Machine Learning workspace

Exam Radar

- Official Blueprint Mapping: Create and manage data assets; Create and manage environments; Create and manage components; Share assets across workspaces by using registries.
- Domain Weight: Design and implement an MLOps infrastructure represents 15-20% of the official skills measured.
- Core Priority: The exam tests whether the candidate can operate this sub-skill as a concrete Azure workflow inside Design and implement an MLOps infrastructure, not merely identify the service name.
- High Frequency: Expect scenario wording that combines Data asset, Environment, Component, and verification evidence from commands, logs, metrics, or portal state.
- Confusion Alert: A local workspace component cannot be reused in another workspace until it is published or referenced through a registry.
- Scenario Logic: Choose the answer that creates a verifiable dependency chain: configure the object, bind identity or version, run the operation, then prove the resulting state.
- Version Delta: AI-300 combines Azure Machine Learning and Microsoft Foundry under AIOps, so this point must be read as an operational platform task rather than a standalone concept.
- Failure Trigger: Pipelines fail when an asset version is omitted, archived, or resolved from the wrong workspace scope.

- Operational Dependency: The task depends on Data asset, Environment, Component, Registry, Model asset, Workspace asset reference and a validation step that proves the configured state is actually usable.
- How the Exam Asks It: A pipeline in a production workspace cannot reuse a component or environment created in a development workspace. The question asks how to make the asset version discoverable and reproducible across workspaces.
- How Distractors Are Designed: Distractors suggest copying source files manually, renaming the component, editing the pipeline display name, or giving users workspace Reader. These do not create a versioned asset contract that another workspace can resolve.
- Why the Correct Answer Works: The correct answer registers or publishes the asset with an explicit version, preferably through a registry for cross-workspace reuse, because pipeline resolution depends on asset name, version, and scope.

Atomic Deconstruction - Operational Level

Microscopic technical focus: Versioning data assets, environments, components, and registry-shared artifacts.

Beginner explanation: An asset is a versioned contract. A file path, Conda file, or script folder becomes exam-relevant only when Azure ML can resolve it by name, version, and scope.

Operational split for this point: start with Data asset, then verify Environment and Component before trusting any production outcome. The exam is testing whether the candidate can locate the missing dependency, not whether the candidate recognizes every service name in the scenario.

For this knowledge point, the target objects are Data asset, Environment, Component, Registry, Model asset, Workspace asset reference. The exam usually describes one broken link in that chain. The correct answer is the option that restores the missing operational dependency rather than the option that only describes the platform at a high level.

Why-layer: Data asset becomes exam-relevant only when the surrounding dependency chain can run. In this topic, Pipelines fail when an asset version is omitted, archived, or resolved from the wrong workspace scope. The correct configuration matters because it changes the state that controls execution, authorization, resolution, evaluation, or observability; a nearby but unrelated action leaves the same failure mode in place.

Decision tree: if the scenario describes access failure, inspect identity and RBAC before changing compute or code; if it describes unresolved assets, inspect name, version, and scope; if it describes runtime failure, inspect logs, endpoint invocation, metrics, or evaluation output; if it describes quality degradation, inspect data, retrieval, evaluation, and monitoring evidence before changing the model.

Common mistakes: Selecting a familiar Azure service without checking the missing dependency in the scenario. Treating a successful create operation as proof of runtime behavior. Choosing a monitoring action when the scenario asks for configuration or access remediation.

Practice question: A pipeline in a production workspace cannot reuse a component or environment created in a development workspace. The question asks how to make the asset version discoverable and reproducible

across workspaces.
A. Publish the component or environment as a versioned asset in an Azure ML registry and reference that registry version from the production pipeline.
B. Copy the component YAML file into the production repository and keep the same display name.
C. Give the production workspace Reader access to the development workspace.
D. Rename the pipeline job so it matches the development job name.

expected answer: A

Explanation: A is correct because cross-workspace reuse depends on asset name, version, and registry scope. B copies text but not a registered asset, C does not create a resolvable component contract, and D changes only metadata.

The common decision point is: A local workspace component cannot be reused in another workspace until it is published or referenced through a registry. Therefore, read every scenario for the actor, the resource scope, the object version, the network path, the metric threshold, and the expected observable result.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Data asset | Version | Immutable named version or latest label | Unversioned local path until created | Datastore or URI path | Pipeline input resolves to stale or missing data |

| Environment | Image and dependencies | Curated image, custom image, Conda specification | Draft YAML before registration | ACR/base image/package feed | Job fails during image build or import |
| Component | Interface contract | Inputs, outputs, command, environment | Local YAML until registered | Workspace or registry scope | Pipeline compilation cannot resolve component |

| Registry asset | Sharing scope | Registry name plus asset name/version | Unavailable outside workspace | Registry permissions and region support | Production workspace cannot consume approved asset |

Step-by-Step Execution Path

1. Execute the operational step.
   az ml environment create --file environment.yml -g rg-ai300 -w mlw-ai300-dev
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Register the environment first because component execution must reference a reproducible runtime image.

Checkpoint: Environment list shows the expected name and version.

2. Execute the operational step.
   az ml component create --file component.yml -g rg-ai300 -w mlw-ai300-dev

Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Create the component after its environment exists so the component contract can resolve its runtime dependency.

Checkpoint: Component show output contains inputs, outputs, command, and environment reference.

3. Execute the operational step.
   az ml component create --file component.yml --registry-name ai300registry
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Publish to registry when another workspace must consume the same component without copying YAML by hand.

Checkpoint: Registry component list shows the approved version.

4. Execute the operational step.
   az ml job create --file pipeline.yml -g rg-ai300-prod -w mlw-ai300-prod
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Run the pipeline in the consuming workspace to prove registry-scoped asset resolution works.

Checkpoint: Pipeline job graph resolves the registry component version and starts execution.

Technical Chain

A user, workflow, or deployment command targets Data asset and submits configuration to Azure control plane or a project runtime. Azure validates identity, resource scope, quota, version references, and network reachability because the runtime cannot safely use an object that is not authorized, versioned, reachable, or measurable. The configured object then participates in the runtime path through Environment, Component, Registry. This sequence works because each object unlocks the next dependency: identity allows access, versioning allows reproducibility, network resolution allows execution, and telemetry allows verification. When the workload executes, telemetry, status output, logs, API response, or evaluation metrics prove whether the chain is complete. If the chain breaks, the failure appears as the operational symptom described in the scenario: Pipelines fail when an asset version is omitted, archived, or resolved from the wrong workspace scope. An incorrect configuration creates the observed failure because it changes a nearby object while leaving the actual missing dependency unresolved.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

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

| List registered environment version | `az ml environment show -n <env-name> --version <version> -g rg-ai300 -w mlw-ai300-dev` | Environment resolves with expected image and dependencies. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

| Inspect component contract | `az ml component show -n <component-name> --version <version> -g rg-ai300 -w mlw-ai300-dev` | Inputs, outputs, command, and environment reference are present. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |
| Verify registry asset | `az ml component show -n <component-name> --version <version> --registry-name ai300registry` | Registry returns the shared component version. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

| Confirm consuming pipeline state | `az ml job show --name <pipeline-job> -g rg-ai300-prod -w mlw-ai300-prod --query status` | Pipeline compiles and starts with registry asset references. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

Implement infrastructure as code for Azure Machine Learning

Exam Radar

- Official Blueprint Mapping: Configure GitHub integration with Machine Learning to enable secure access; Deploy Machine Learning workspaces and resources by using Bicep and Azure CLI; Automate resource provisioning by using GitHub Actions workflows; Restrict network access to Machine Learning workspaces; Manage source control for machine learning projects by using Git.
- Domain Weight: Design and implement an MLOps infrastructure represents 15-20% of the official skills measured.
- Core Priority: The exam tests whether the candidate can operate this sub-skill as a concrete Azure workflow inside Design and implement an MLOps infrastructure, not merely identify the service name.
- High Frequency: Expect scenario wording that combines Bicep module, GitHub Actions workflow, Federated credential, and verification evidence from commands, logs, metrics, or portal state.
- Confusion Alert: Secret-based deployment can work but creates rotation and leakage risk; OIDC federation provides short-lived workflow authentication.
- Scenario Logic: Choose the answer that creates a verifiable dependency chain: configure the object, bind identity or version, run the operation, then prove the resulting state.
- Version Delta: AI-300 combines Azure Machine Learning and Microsoft Foundry under AIOps, so this point must be read as an operational platform task rather than a standalone concept.
- Failure Trigger: Automation fails before resource deployment when the workflow lacks id-token permission or the federated credential subject does not match the branch.
- Operational Dependency: The task depends on Bicep module, GitHub Actions workflow, Federated credential, Azure CLI deployment, Network rule, Git repository and a validation step that proves the configured state is actually usable.
- How the Exam Asks It: A team needs repeatable Azure ML environments and wants GitHub Actions to deploy workspaces, storage, network rules, and compute without storing long-lived secrets.

- How Distractors Are Designed: Distractors use manual portal creation, broad subscription Owner secrets, post-deployment screenshots, or local CLI commands from an engineer laptop. These approaches cannot prove repeatability or secure pipeline identity.
- Why the Correct Answer Works: The correct answer uses Bicep or ARM plus GitHub Actions OIDC/federated credentials and validates deployment outputs, because the infrastructure state must be reproducible and attributable to CI.

Atomic Deconstruction - Operational Level

Microscopic technical focus: Deploying secure Machine Learning infrastructure with Bicep, Azure CLI, GitHub Actions, and source control.

Beginner explanation: IaC means the environment can be recreated by a pipeline. The exam cares less about the template language itself and more about whether identity, network, and resource dependencies are repeatable.

Operational split for this point: start with Bicep module, then verify GitHub Actions workflow and Federated credential before trusting any production outcome. The exam is testing whether the candidate can locate the missing dependency, not whether the candidate recognizes every service name in the scenario.

For this knowledge point, the target objects are Bicep module, GitHub Actions workflow, Federated credential, Azure CLI deployment, Network rule, Git repository. The exam usually describes one broken link in that chain. The correct answer is the option that restores the missing operational dependency rather than the option that only describes the platform at a high level.

Why-layer: Bicep module becomes exam-relevant only when the surrounding dependency chain can run. In this topic, Automation fails before resource deployment when the workflow lacks id-token permission or the federated credential subject does not match the branch. The correct configuration matters because it changes the state that controls execution, authorization, resolution, evaluation, or observability; a nearby but unrelated action leaves the same failure mode in place.

Decision tree: if the scenario describes access failure, inspect identity and RBAC before changing compute or code; if it describes unresolved assets, inspect name, version, and scope; if it describes runtime failure, inspect logs, endpoint invocation, metrics, or evaluation output; if it describes quality degradation, inspect data, retrieval, evaluation, and monitoring evidence before changing the model.

Common mistakes: Selecting a familiar Azure service without checking the missing dependency in the scenario. Treating a successful create operation as proof of runtime behavior. Choosing a monitoring action when the scenario asks for configuration or access remediation.

Practice question: A team needs repeatable Azure ML environments and wants GitHub Actions to deploy workspaces, storage, network rules, and compute without storing long-lived secrets.
A. Use Bicep or ARM deployment from GitHub Actions with OIDC federation and validate the resource-group deployment output.
B. Create the workspace manually in the portal and export screenshots for audit evidence.

C. Store an Owner client secret in GitHub repository secrets and reuse it for every environment.
D. Run Azure CLI commands locally from an engineer workstation after each merge.

expected answer: A

Explanation: A is correct because it provides repeatable infrastructure and short-lived CI identity. B and D are not repeatable pipeline controls, while C works technically but creates long-lived credential risk.

The common decision point is: Secret-based deployment can work but creates rotation and leakage risk; OIDC federation provides short-lived workflow authentication. Therefore, read every scenario for the actor, the resource scope, the object version, the network path, the metric threshold, and the expected observable result.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Bicep module | Resource graph | Workspace, storage, key vault, ACR, networking | Template only until deployed | Resource group and provider registration | Deployment fails or creates incomplete dependency graph |

| Federated credential | Subject claim | Repository, branch, environment, or workflow subject | No trust relationship | Entra app and GitHub OIDC token | azure/login fails before deployment |
| GitHub workflow | Permissions | id-token: write and contents: read | Token unavailable by default | OIDC federation and Azure login | Workflow cannot request Azure token |

| Deployment output | State evidence | Succeeded, failed, outputs JSON | No evidence until queried | Azure Resource Manager deployment record | Pipeline cannot prove created resources |

Step-by-Step Execution Path

1. Execute the operational step.
   az deployment group validate -g rg-ai300 -f infra/main.bicep
   Command type: Azure CLI verification; confirm parameters against the active Azure CLI version.

Reason: Validate before deployment because syntax and parameter errors should fail before any resource mutation occurs.

Checkpoint: Validation returns no template errors.

2. Execute the operational step.
   az deployment group create -g rg-ai300 -f infra/main.bicep
   Command type: Azure CLI verification; confirm parameters against the active Azure CLI version.

Reason: Deploy the full dependency graph together so workspace resources and dependent services remain consistent.

Checkpoint: Deployment provisioningState is Succeeded.

3. Execute the operational step.
   az ad app federated-credential create --id --parameters credential.json
   Command type: Azure CLI verification; confirm parameters against the active Azure CLI version.

Reason: Create OIDC trust so GitHub can obtain short-lived Azure tokens without storing a client secret.

Checkpoint: The federated credential subject matches the repository branch or environment.

4. Execute the operational step.
   gh run view --log
   Command type: GitHub CLI workflow verification.

Reason: Inspect workflow logs because CI evidence must show the template was deployed by the pipeline identity.

Checkpoint: Logs show azure/login and deployment steps completed successfully.

Technical Chain

A user, workflow, or deployment command targets Bicep module and submits configuration to Azure control plane or a project runtime. Azure validates identity, resource scope, quota, version references, and network reachability because the runtime cannot safely use an object that is not authorized, versioned, reachable, or measurable. The configured object then participates in the runtime path through GitHub Actions workflow, Federated credential, Azure CLI deployment. This sequence works because each object unlocks the next dependency: identity allows access, versioning allows reproducibility, network resolution allows execution, and telemetry allows verification. When the workload executes, telemetry, status output, logs, API response, or evaluation metrics prove whether the chain is complete. If the chain breaks, the failure appears as the operational symptom described in the scenario: Automation fails before resource deployment when the workflow lacks id-token permission or the federated credential subject does not match the branch. An incorrect configuration creates the observed failure because it changes a nearby object while leaving the actual missing dependency unresolved.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

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

| Validate template syntax | `az deployment group validate -g rg-ai300 -f infra/main.bicep` | Validation completes without template errors. Command type: Azure CLI verification; confirm parameters against the active Azure CLI version. |

| Inspect deployment state | `az deployment group show -g rg-ai300 -n main --query properties.provisioningState` | Provisioning state is Succeeded. Command type: Azure CLI verification;

confirm parameters against the active Azure CLI version. |
| Verify federated credential | `az ad app federated-credential list --id <app-id> -o table` | Credential subject matches GitHub repository branch or environment. Command type: Azure CLI verification; confirm parameters against the active Azure CLI version. |

| Inspect workflow evidence | `gh run view <run-id> --log` | Log shows azure/login and deployment steps succeeded. Command type: GitHub CLI workflow verification. |

Practice Questions

1. A team is creating an Azure Machine Learning workspace for regulated model development. The workspace must restrict public ingress and allow notebooks and jobs to access storage through private paths. Which design should be prioritized?
   A. Create the workspace with a public endpoint and rely on workspace-level tags for access control
   B. Enable a private endpoint for the workspace and connect dependent storage and container registry through private networking
   C. Deploy only a larger compute cluster and disable experiment tracking
   D. Store training data in a local developer machine and upload model outputs manually

2. An Azure Machine Learning workspace deployment succeeds, but pipeline jobs fail when reading datasets from the default datastore. What should the engineer verify first?
   A. Whether the datastore credential or managed identity has the required Storage Blob Data access
   B. Whether the model registry contains at least two model versions
   C. Whether the prompt template uses a lower temperature setting
   D. Whether the endpoint has autoscale enabled

3. A platform team wants repeatable Azure Machine Learning workspace creation across development, test, and production. Which approach best supports environment consistency and reviewable changes?
   A. Manually create each workspace in the portal and document the screenshots
   B. Export experiment metrics from MLflow after every run
   C. Ask each data scientist to create a personal workspace and share the endpoint URL
   D. Use Bicep or Terraform templates stored in source control with parameter files per environment

4. A data science team needs to share curated feature data and trained model artifacts between projects in Azure Machine Learning. Which workspace concept should be used?
   A. Azure Policy exemptions only
   B. Application Insights availability tests only
   C. Assets such as data assets, environments, components, and models
   D. Prompt flow variants only

5. A pipeline run fails because the training script cannot reproduce a previous environment even though the code is unchanged. What should be versioned in Azure Machine Learning to reduce this

risk?

A. The workspace display name

B. The compute cluster minimum node count only

C. The environment definition, including base image and package dependencies

D. The Application Insights sampling percentage only

6. An organization wants Azure Machine Learning compute to scale down when idle but still support queued training jobs. Which resource should be configured?

   A. A compute cluster with appropriate min nodes, max nodes, and idle scale-down settings

   B. A managed online endpoint with blue-green traffic splitting

   C. A Key Vault access policy for secret rotation

   D. An Azure AI Search semantic configuration

7. A workspace template includes Azure Machine Learning, storage, Key Vault, and Application Insights. Deployment succeeds, but training jobs cannot pull container images. Which dependency is most likely missing or misconfigured?

   A. A model evaluation threshold

   B. A semantic ranker configuration

   C. A prompt version tag

   D. Azure Container Registry access or network connectivity for the workspace compute

8. A team wants to prevent unapproved users from creating expensive GPU compute in Azure Machine Learning. Which control is most appropriate?

   A. Increase endpoint request timeout values

   B. Apply Azure RBAC and policy controls that restrict compute creation and SKU usage

   C. Lower model temperature to zero

   D. Add more labels to registered models

9. After migrating an Azure Machine Learning workspace to private networking, users can open the workspace but jobs fail when accessing Key Vault secrets. What should be checked first?

   A. The model endpoint traffic percentage

   B. The number of examples in the prompt

   C. The Key Vault private endpoint, firewall rules, and managed identity permissions

   D. The Azure AI Search vector dimension

10. A compliance reviewer asks how the team can prove that workspace infrastructure changes were approved before deployment. What evidence best supports this?

    A. A list of prompt template names

    B. A single screenshot of the Azure portal overview page

    C. The most recent model prediction response

    D. Pull requests and deployment logs for the infrastructure-as-code repository

Implement machine learning model lifecycle and operations

Orchestrate model training in Azure Machine Learning

Exam Radar

- Official Blueprint Mapping: Configure experiment tracking with MLflow; Use automated machine learning to explore optimal models; Use notebooks for experimentation and exploration; Automate hyperparameter tuning; Run model training scripts; Manage distributed training for large and deep learning models; Implement training pipelines; Compare model performance across jobs.
- Domain Weight: Implement machine learning model lifecycle and operations represents 25-30% of the official skills measured.
- Core Priority: The exam tests whether the candidate can operate this sub-skill as a concrete Azure workflow inside Implement machine learning model lifecycle and operations, not merely identify the service name.
- High Frequency: Expect scenario wording that combines Command job, MLflow run, AutoML job, and verification evidence from commands, logs, metrics, or portal state.
- Confusion Alert: A notebook experiment is useful for exploration but a command job or pipeline is required for repeatable lifecycle automation.
- Scenario Logic: Choose the answer that creates a verifiable dependency chain: configure the object, bind identity or version, run the operation, then prove the resulting state.
- Version Delta: AI-300 combines Azure Machine Learning and Microsoft Foundry under AIOps, so this point must be read as an operational platform task rather than a standalone concept.
- Failure Trigger: Training output cannot be promoted when MLflow metrics, input data versions, or artifact paths are not captured.
- Operational Dependency: The task depends on Command job, MLflow run, AutoML job, Sweep job, Pipeline component, Distributed training process and a validation step that proves the configured state is actually usable.
- How the Exam Asks It: A model candidate cannot be compared or promoted because training was run interactively without tracked metrics, input versions, or model artifacts.
- How Distractors Are Designed: Distractors recommend changing VM size, exporting notebook output, or emailing metric screenshots. These may help experimentation but do not create auditable training lineage.
- Why the Correct Answer Works: The correct answer submits a tracked command, sweep, AutoML, distributed, or pipeline job with MLflow logging and versioned inputs because promotion depends on reproducible run evidence.

Atomic Deconstruction - Operational Level

Microscopic technical focus: Running tracked command jobs, AutoML jobs, hyperparameter sweeps, distributed training, and pipeline jobs.

Beginner explanation: A training job is the audited version of running Python. It captures code, inputs, environment, compute, metrics, and artifacts so the result can be compared and promoted.

Operational split for this point: start with Command job, then verify MLflow run and AutoML job before trusting any production outcome. The exam is testing whether the candidate can locate the missing dependency, not whether the candidate recognizes every service name in the scenario.

For this knowledge point, the target objects are Command job, MLflow run, AutoML job, Sweep job, Pipeline component, Distributed training process. The exam usually describes one broken link in that chain. The correct answer is the option that restores the missing operational dependency rather than the option that only describes the platform at a high level.

Why-layer: Command job becomes exam-relevant only when the surrounding dependency chain can run. In this topic, Training output cannot be promoted when MLflow metrics, input data versions, or artifact paths are not captured. The correct configuration matters because it changes the state that controls execution, authorization, resolution, evaluation, or observability; a nearby but unrelated action leaves the same failure mode in place.

Decision tree: if the scenario describes access failure, inspect identity and RBAC before changing compute or code; if it describes unresolved assets, inspect name, version, and scope; if it describes runtime failure, inspect logs, endpoint invocation, metrics, or evaluation output; if it describes quality degradation, inspect data, retrieval, evaluation, and monitoring evidence before changing the model.

Common mistakes: Selecting a familiar Azure service without checking the missing dependency in the scenario. Treating a successful create operation as proof of runtime behavior. Choosing a monitoring action when the scenario asks for configuration or access remediation.

Practice question: A model candidate cannot be compared or promoted because training was run interactively without tracked metrics, input versions, or model artifacts.
A. Submit training as an Azure ML job or pipeline with MLflow metrics, versioned inputs, environment, compute, and artifact outputs.
B. Run the notebook manually and copy the final accuracy value into the release notes.
C. Increase the VM size used by the notebook kernel to reduce execution time.
D. Export the trained model folder to a shared drive without job metadata.

expected answer: A

Explanation: A is correct because promotion requires tracked lineage and comparable metrics. B and D lose auditable run evidence, while C changes performance but not lifecycle traceability.

The common decision point is: A notebook experiment is useful for exploration but a command job or pipeline is required for repeatable lifecycle automation. Therefore, read every scenario for the actor, the resource

scope, the object version, the network path, the metric threshold, and the expected observable result.

Component Specifications

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

| ---------- | --------- | ------------------------------------- | ---------------------------- | ------------------------- | -- |
| Command job | Code and command | Script path plus command string | Draft YAML until submitted | Compute, environment, inputs | Job cannot reproduce notebook-only training |

| MLflow run | Logged evidence | Parameters, metrics, artifacts, model path | Empty until instrumented | Training script logging | No objective comparison across candidates |
| Sweep job | Search space | Choice, uniform, loguniform, grid | No trials until submitted | Primary metric and limits | Tuning produces unbounded or unranked trials |

| Pipeline job | Step dependency | DAG inputs and outputs | No orchestration until submitted | Component contracts | Registration step cannot consume training output |

Step-by-Step Execution Path

1. Execute the operational step.
   az ml job create --file train-job.yml -g rg-ai300 -w mlw-ai300-dev
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Submit training as a job because Azure ML can track inputs, code, environment, metrics, and artifacts only inside a managed run.

Checkpoint: Job status reaches Completed.

2. Execute the operational step.
   az ml job stream --name -g rg-ai300 -w mlw-ai300-dev
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Stream logs during execution to catch package, data, or script failures before promotion steps consume bad output.

Checkpoint: Logs show training completed and metrics were logged.

3. Execute the operational step.
   az ml job show --name --query outputs -g rg-ai300 -w mlw-ai300-dev
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Inspect outputs because model registration needs the exact artifact URI, not a local notebook path.

Checkpoint: Output contains a model artifact path.

4. Execute the operational step.
   az ml job create --file train-pipeline.yml -g rg-ai300 -w mlw-ai300-dev

Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Use a pipeline when training, evaluation, and registration must run as a dependent graph.

Checkpoint: Pipeline graph shows completed train and evaluation nodes.

Technical Chain

A user, workflow, or deployment command targets Command job and submits configuration to Azure control plane or a project runtime. Azure validates identity, resource scope, quota, version references, and network reachability because the runtime cannot safely use an object that is not authorized, versioned, reachable, or measurable. The configured object then participates in the runtime path through MLflow run, AutoML job, Sweep job. This sequence works because each object unlocks the next dependency: identity allows access, versioning allows reproducibility, network resolution allows execution, and telemetry allows verification. When the workload executes, telemetry, status output, logs, API response, or evaluation metrics prove whether the chain is complete. If the chain breaks, the failure appears as the operational symptom described in the scenario: Training output cannot be promoted when MLflow metrics, input data versions, or artifact paths are not captured. An incorrect configuration creates the observed failure because it changes a nearby object while leaving the actual missing dependency unresolved.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

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

| Confirm job completion | `az ml job show --name <job-name> -g rg-ai300 -w mlw-ai300-dev --query status` | Status is Completed. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

| Read training logs | `az ml job stream --name <job-name> -g rg-ai300 -w mlw-ai300-dev` | Logs show successful training and metric logging. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

| Inspect job outputs | `az ml job show --name <job-name> -g rg-ai300 -w mlw-ai300-dev --query outputs` | Model artifact output path is present. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

| Compare runs | `az ml job list -g rg-ai300 -w mlw-ai300-dev --query "[].{name:name,status:status}"` | Candidate jobs are visible for comparison. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

Implement model registration and versioning

Exam Radar

- Official Blueprint Mapping: Package a feature retrieval specification with the model artifact; Register an MLflow model; Evaluate a model by using responsible AI principles; Manage model lifecycle, including archiving models.
- Domain Weight: Implement machine learning model lifecycle and operations represents 25-30% of the official skills measured.
- Core Priority: The exam tests whether the candidate can operate this sub-skill as a concrete Azure workflow inside Implement machine learning model lifecycle and operations, not merely identify the service name.
- High Frequency: Expect scenario wording that combines Registered model, MLflow artifact, Feature retrieval specification, and verification evidence from commands, logs, metrics, or portal state.
- Confusion Alert: A job artifact is not an auditable production model until registration stores name, version, path, tags, and lineage.
- Scenario Logic: Choose the answer that creates a verifiable dependency chain: configure the object, bind identity or version, run the operation, then prove the resulting state.
- Version Delta: AI-300 combines Azure Machine Learning and Microsoft Foundry under AIOps, so this point must be read as an operational platform task rather than a standalone concept.
- Failure Trigger: Deployment cannot resolve the model when the registered version points to a missing artifact or incompatible feature retrieval contract.
- Operational Dependency: The task depends on Registered model, MLflow artifact, Feature retrieval specification, Model version, Responsible AI report, Archived model and a validation step that proves the configured state is actually usable.
- How the Exam Asks It: A deployment pipeline needs to promote the exact model that passed evaluation while preserving lineage, feature assumptions, and rollback options.
- How Distractors Are Designed: Distractors deploy directly from a local folder, overwrite the same model name without version tags, or rely on the latest run metric in Studio. These lose reproducible promotion evidence.
- Why the Correct Answer Works: The correct answer registers a model version from the job artifact with tags/properties and uses that version in deployment, because production must reference an immutable model asset.

Atomic Deconstruction - Operational Level

Microscopic technical focus: Packaging feature specifications, registering MLflow models, applying responsible AI evidence, and archiving versions.

Beginner explanation: Registration turns a training output into a named production candidate. Without a model version, an endpoint cannot reliably point to the exact artifact that passed evaluation.

Operational split for this point: start with Registered model, then verify MLflow artifact and Feature retrieval specification before trusting any production outcome. The exam is testing whether the candidate can locate the missing dependency, not whether the candidate recognizes every service name in the scenario.

For this knowledge point, the target objects are Registered model, MLflow artifact, Feature retrieval specification, Model version, Responsible AI report, Archived model. The exam usually describes one broken link in that chain. The correct answer is the option that restores the missing operational dependency rather than the option that only describes the platform at a high level.

Why-layer: Registered model becomes exam-relevant only when the surrounding dependency chain can run. In this topic, Deployment cannot resolve the model when the registered version points to a missing artifact or incompatible feature retrieval contract. The correct configuration matters because it changes the state that controls execution, authorization, resolution, evaluation, or observability; a nearby but unrelated action leaves the same failure mode in place.

Decision tree: if the scenario describes access failure, inspect identity and RBAC before changing compute or code; if it describes unresolved assets, inspect name, version, and scope; if it describes runtime failure, inspect logs, endpoint invocation, metrics, or evaluation output; if it describes quality degradation, inspect data, retrieval, evaluation, and monitoring evidence before changing the model.

Common mistakes: Selecting a familiar Azure service without checking the missing dependency in the scenario. Treating a successful create operation as proof of runtime behavior. Choosing a monitoring action when the scenario asks for configuration or access remediation.

Practice question: A deployment pipeline needs to promote the exact model that passed evaluation while preserving lineage, feature assumptions, and rollback options.
A. Register the MLflow model from the approved job artifact with an explicit model version and promotion tags.
B. Deploy the latest local model folder directly to the endpoint.
C. Overwrite the same model name every time a better run appears.
D. Use the highest metric shown in Studio without registering the artifact.

expected answer: A

Explanation: A is correct because production needs an immutable version tied to lineage. B bypasses registry control, C destroys version history, and D identifies a run but not a deployable model contract.

The common decision point is: A job artifact is not an auditable production model until registration stores name, version, path, tags, and lineage. Therefore, read every scenario for the actor, the resource scope, the object version, the network path, the metric threshold, and the expected observable result.

Component Specifications

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

| --- | --- | --- | --- | --- | --- |
| Registered model | Name and version | Integer version or label | Absent until created | Job artifact or model path | Endpoint cannot resolve a stable artifact |

| MLflow artifact | Model flavor | MLflow model directory | Training output only | MLmodel file and dependencies | Container cannot load model signature |
| Feature spec | Input contract | Feature names, types, transformations | Implicit unless documented | Training and scoring parity | Inference receives incompatible feature shape |

| Model tag | Promotion metadata | stage, owner, metric, dataset | No governance metadata | Release pipeline convention | Rollback cannot identify approved candidate |

Step-by-Step Execution Path

1. Execute the operational step.
   az ml model create --name churn-model --type mlflow_model --path azureml://jobs//outputs/artifacts/paths/model
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Register from the job artifact so the model version points to tracked training lineage rather than a developer machine.

Checkpoint: Model show returns name, version, path, and type.

2. Execute the operational step.
   az ml model update --name churn-model --version 1 --set tags.stage=staging tags.metric_auc=0.91
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Add tags because release gates and rollback decisions need searchable promotion metadata.

Checkpoint: Model metadata contains stage and metric tags.

3. Execute the operational step.
   az ml model show --name churn-model --version 1 --query path
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Verify the artifact path before deployment so the endpoint does not pull an unintended or missing model version.

Checkpoint: Path references the expected training job output.

4. Execute the operational step.
   az ml model archive --name churn-model --version 0
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Archive rejected or superseded versions so production pipelines do not accidentally select obsolete assets.

Checkpoint: Archived version no longer appears as active in model list.

Technical Chain

A user, workflow, or deployment command targets Registered model and submits configuration to Azure control plane or a project runtime. Azure validates identity, resource scope, quota, version references, and network reachability because the runtime cannot safely use an object that is not authorized, versioned, reachable, or measurable. The configured object then participates in the runtime path through MLflow artifact, Feature retrieval specification, Model version. This sequence works because each object unlocks the next dependency: identity allows access, versioning allows reproducibility, network resolution allows execution, and telemetry allows verification. When the workload executes, telemetry, status output, logs, API response, or evaluation metrics prove whether the chain is complete. If the chain breaks, the failure appears as the operational symptom described in the scenario: Deployment cannot resolve the model when the registered version points to a missing artifact or incompatible feature retrieval contract. An incorrect configuration creates the observed failure because it changes a nearby object while leaving the actual missing dependency unresolved.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

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

| Inspect registered model | `az ml model show --name churn-model --version 1 -g rg-ai300 -w mlw-ai300-dev` | Model type, path, and version are returned. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

| Verify model tags | `az ml model show --name churn-model --version 1 --query tags -g rg-ai300 -w mlw-ai300-dev` | Stage and metric tags are present. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

| Confirm artifact lineage | `az ml model show --name churn-model --version 1 --query path -g rg-ai300 -w mlw-ai300-dev` | Path references the approved training job output. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

| Check archived state | `az ml model list --name churn-model -g rg-ai300 -w mlw-ai300-dev -o table` | Archived versions are not selected for active promotion. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

Deploy machine learning models for production environments

Exam Radar

- Official Blueprint Mapping: Deploy models as real-time or batch endpoints with managed inference options; Test and troubleshoot model endpoints; Implement progressive rollout and safe rollback strategies.
- Domain Weight: Implement machine learning model lifecycle and operations represents 25-30% of the official skills measured.
- Core Priority: The exam tests whether the candidate can operate this sub-skill as a concrete Azure workflow inside Implement machine learning model lifecycle and operations, not merely identify the service name.
- High Frequency: Expect scenario wording that combines Online endpoint, Online deployment, Batch endpoint, and verification evidence from commands, logs, metrics, or portal state.
- Confusion Alert: A successful deployment operation does not prove production readiness until invocation, logs, traffic split, and latency are verified.
- Scenario Logic: Choose the answer that creates a verifiable dependency chain: configure the object, bind identity or version, run the operation, then prove the resulting state.
- Version Delta: AI-300 combines Azure Machine Learning and Microsoft Foundry under AIOps, so this point must be read as an operational platform task rather than a standalone concept.
- Failure Trigger: Requests fail when scoring schema, environment dependencies, model loading, authentication mode, or traffic routing is wrong.
- Operational Dependency: The task depends on Online endpoint, Online deployment, Batch endpoint, Scoring script, Traffic allocation, Inference environment and a validation step that proves the configured state is actually usable.
- How the Exam Asks It: A team has a registered model and must expose it for online or batch scoring with safe rollout and rollback controls.
- How Distractors Are Designed: Distractors stop after model registration, change training code, increase batch size for online latency, or assign 100 percent traffic to an untested deployment. These ignore serving behavior.
- Why the Correct Answer Works: The correct answer creates the proper endpoint and deployment, invokes it with a sample request, inspects logs, and shifts traffic gradually because deployment success alone does not prove scoring readiness.

Atomic Deconstruction - Operational Level

Microscopic technical focus: Configuring real-time endpoints, batch endpoints, managed inference, progressive rollout, and rollback.

Beginner explanation: Deployment is where a model becomes callable. The exam separates model existence from scoring readiness, traffic routing, and rollback control.

Operational split for this point: start with Online endpoint, then verify Online deployment and Batch endpoint before trusting any production outcome. The exam is testing whether the candidate can locate the missing dependency, not whether the candidate recognizes every service name in the scenario.

For this knowledge point, the target objects are Online endpoint, Online deployment, Batch endpoint, Scoring script, Traffic allocation, Inference environment. The exam usually describes one broken link in that chain. The correct answer is the option that restores the missing operational dependency rather than the option that only describes the platform at a high level.

Why-layer: Online endpoint becomes exam-relevant only when the surrounding dependency chain can run. In this topic, Requests fail when scoring schema, environment dependencies, model loading, authentication mode, or traffic routing is wrong. The correct configuration matters because it changes the state that controls execution, authorization, resolution, evaluation, or observability; a nearby but unrelated action leaves the same failure mode in place.

Decision tree: if the scenario describes access failure, inspect identity and RBAC before changing compute or code; if it describes unresolved assets, inspect name, version, and scope; if it describes runtime failure, inspect logs, endpoint invocation, metrics, or evaluation output; if it describes quality degradation, inspect data, retrieval, evaluation, and monitoring evidence before changing the model.

Common mistakes: Assuming model registration means the model is callable. Skipping endpoint invocation before shifting production traffic. Using batch endpoint logic for low-latency online scoring.

Practice question: A team has a registered model and must expose it for online or batch scoring with safe rollout and rollback controls.
A. Create an endpoint deployment, invoke it with a sample request, inspect logs, and shift traffic gradually.
B. Register the model and assume it is available for real-time scoring.
C. Assign 100 percent production traffic to the new deployment immediately after creation.
D. Increase the training batch size to improve online inference latency.

expected answer: A

Explanation: A is correct because endpoint readiness requires serving validation and traffic control. B stops before serving, C removes safe rollout protection, and D confuses training configuration with inference behavior.

The common decision point is: A successful deployment operation does not prove production readiness until invocation, logs, traffic split, and latency are verified. Therefore, read every scenario for the actor, the resource scope, the object version, the network path, the metric threshold, and the expected observable result.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Online endpoint | Auth and traffic | Key, AML token, managed identity; 0-100 percent split | Endpoint has no scoring until deployment exists | Deployment and scoring route | Requests return 404, 401, or no healthy deployment |

| Online deployment | Model/runtime binding | Model version, environment, code, instance type | Unhealthy until provisioned | Endpoint and model asset | Container fails to start or load model |
| Batch endpoint | Input/output binding | URI file, folder, data asset | No scoring job until invoked | Batch deployment and compute | Batch job fails or writes incomplete output |

| Traffic split | Production routing | blue/green percentage | No traffic until assigned | Healthy deployments | Users hit unvalidated version or rollback fails |

Step-by-Step Execution Path

1. Execute the operational step.
   az ml online-endpoint create --file endpoint.yml -g rg-ai300 -w mlw-ai300-prod
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Create the endpoint first because deployments need a stable scoring URL and authentication boundary.

Checkpoint: Endpoint provisioning state is Succeeded.

2. Execute the operational step.
   az ml online-deployment create --file blue-deployment.yml -g rg-ai300 -w mlw-ai300-prod
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Bind model, environment, scoring script, and instance type into a deployment so the endpoint has a runnable container.

Checkpoint: Deployment state is Healthy or Succeeded.

3. Execute the operational step.
   az ml online-endpoint invoke --name churn-endpoint --request-file sample.json -g rg-ai300 -w mlw-ai300-prod
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Invoke before routing production traffic because container startup does not prove request schema or scoring logic works.

Checkpoint: Response matches the expected prediction schema.

4. Execute the operational step.
   az ml online-endpoint update --name churn-endpoint --traffic blue=90 green=10 -g rg-ai300 -w mlw-ai300-prod
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Shift traffic gradually so metrics and logs can validate the new version before full cutover.

Checkpoint: Endpoint traffic table shows the intended split.

Technical Chain

A user, workflow, or deployment command targets Online endpoint and submits configuration to Azure control plane or a project runtime. Azure validates identity, resource scope, quota, version references, and network reachability because the runtime cannot safely use an object that is not authorized, versioned, reachable, or measurable. The configured object then participates in the runtime path through Online deployment, Batch endpoint, Scoring script. This sequence works because each object unlocks the next dependency: identity allows access, versioning allows reproducibility, network resolution allows execution, and telemetry allows verification. When the workload executes, telemetry, status output, logs, API response, or evaluation metrics prove whether the chain is complete. If the chain breaks, the failure appears as the operational symptom described in the scenario: Requests fail when scoring schema, environment dependencies, model loading, authentication mode, or traffic routing is wrong. An incorrect configuration creates the observed failure because it changes a nearby object while leaving the actual missing dependency unresolved.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Check endpoint state | `az ml online-endpoint show --name churn-endpoint -g rg-ai300 -w mlw-ai300-prod --query provisioning_state` | Endpoint provisioning state is Succeeded. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

| Inspect deployment health | `az ml online-deployment show --endpoint-name churn-endpoint --name blue -g rg-ai300 -w mlw-ai300-prod --query provisioning_state` | Deployment state indicates successful provisioning. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |
| Invoke scoring path | `az ml online-endpoint invoke --name churn-endpoint --request-file sample.json -g rg-ai300 -w mlw-ai300-prod` | Response matches expected scoring schema. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

| Verify traffic allocation | `az ml online-endpoint show --name churn-endpoint -g rg-ai300 -w mlw-ai300-prod --query traffic` | Traffic split matches rollout plan. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

Monitor and maintain machine learning models in production

Exam Radar

- Official Blueprint Mapping: Detect and analyze data drift; Monitor performance metrics of models deployed to production; Configure retraining or alert triggers when thresholds are exceeded.
- Domain Weight: Implement machine learning model lifecycle and operations represents 25-30% of the official skills measured.
- Core Priority: The exam tests whether the candidate can operate this sub-skill as a concrete Azure workflow inside Implement machine learning model lifecycle and operations, not merely identify the service name.
- High Frequency: Expect scenario wording that combines Data drift monitor, Model metric, Alert rule, and verification evidence from commands, logs, metrics, or portal state.
- Confusion Alert: A calendar retraining schedule is weaker than threshold-based retraining when production drift and performance degradation are measurable.
- Scenario Logic: Choose the answer that creates a verifiable dependency chain: configure the object, bind identity or version, run the operation, then prove the resulting state.
- Version Delta: AI-300 combines Azure Machine Learning and Microsoft Foundry under AIOps, so this point must be read as an operational platform task rather than a standalone concept.
- Failure Trigger: Bad predictions persist when production telemetry is collected but no alert, threshold, or retraining action is bound to it.
- Operational Dependency: The task depends on Data drift monitor, Model metric, Alert rule, Retraining pipeline, Baseline dataset, Production endpoint log and a validation step that proves the configured state is actually usable.
- How the Exam Asks It: A deployed model shows degraded predictions or data distribution changes, and the team needs a monitored retraining path rather than manual inspection.
- How Distractors Are Designed: Distractors only redeploy the same model, increase instance count, or review endpoint availability. These may address serving health but not drift or model quality.
- Why the Correct Answer Works: The correct answer monitors production data and model metrics against a baseline, configures alerts, and triggers retraining when thresholds are crossed.

Atomic Deconstruction - Operational Level

Microscopic technical focus: Detecting drift, tracking production metrics, configuring alerts, and triggering retraining workflows.

Beginner explanation: Monitoring answers whether the model still behaves correctly after real data changes. Availability metrics alone do not prove prediction quality.

Operational split for this point: start with Data drift monitor, then verify Model metric and Alert rule before trusting any production outcome. The exam is testing whether the candidate can locate the missing

dependency, not whether the candidate recognizes every service name in the scenario.

For this knowledge point, the target objects are Data drift monitor, Model metric, Alert rule, Retraining pipeline, Baseline dataset, Production endpoint log. The exam usually describes one broken link in that chain. The correct answer is the option that restores the missing operational dependency rather than the option that only describes the platform at a high level.

Why-layer: Data drift monitor becomes exam-relevant only when the surrounding dependency chain can run. In this topic, Bad predictions persist when production telemetry is collected but no alert, threshold, or retraining action is bound to it. The correct configuration matters because it changes the state that controls execution, authorization, resolution, evaluation, or observability; a nearby but unrelated action leaves the same failure mode in place.

Decision tree: if the scenario describes access failure, inspect identity and RBAC before changing compute or code; if it describes unresolved assets, inspect name, version, and scope; if it describes runtime failure, inspect logs, endpoint invocation, metrics, or evaluation output; if it describes quality degradation, inspect data, retrieval, evaluation, and monitoring evidence before changing the model.

Common mistakes: Selecting a familiar Azure service without checking the missing dependency in the scenario. Treating a successful create operation as proof of runtime behavior. Choosing a monitoring action when the scenario asks for configuration or access remediation.

Practice question: A deployed model shows degraded predictions or data distribution changes, and the team needs a monitored retraining path rather than manual inspection.
A. Compare production signals against a baseline, configure alert thresholds, and trigger a governed retraining pipeline when thresholds fail.
B. Redeploy the same model version whenever users report worse predictions.
C. Monitor endpoint availability only because HTTP success proves model quality.
D. Scale out the endpoint instances to reduce all quality degradation.

expected answer: A

Explanation: A is correct because drift and quality require baseline comparison and action. B repeats the same model, C ignores prediction quality, and D addresses capacity rather than model behavior.

The common decision point is: A calendar retraining schedule is weaker than threshold-based retraining when production drift and performance degradation are measurable. Therefore, read every scenario for the actor, the resource scope, the object version, the network path, the metric threshold, and the expected observable result.

Component Specifications

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

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

| Baseline dataset | Reference distribution | Training or approved validation sample | No comparison until selected | Production data schema | Drift score has no meaningful reference |

| Model monitor | Signal type | Data drift, prediction drift, performance, latency | Disabled until configured | Telemetry and baseline | Quality degradation remains invisible |
| Alert rule | Threshold | Metric threshold and action group | No notification | Monitor metric source | Operations team misses degradation window |

| Retraining pipeline | Trigger input | New data, baseline, evaluation gate | Manual until automated | Pipeline components and compute | New model is trained without governance gate |

Step-by-Step Execution Path

1. Execute the operational step.
   az ml online-deployment get-logs --endpoint-name churn-endpoint --name blue -g rg-ai300 -w mlw-ai300-prod
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Read deployment logs first to separate scoring errors from model-quality degradation.

Checkpoint: Logs show whether requests are failing or predictions are merely poor.

2. Execute the operational step.
   az monitor metrics list --resource --metric Requests,Latency
   Command type: Azure Monitor CLI verification; confirm metric names for the selected Azure resource type.

Reason: Check serving metrics because infrastructure instability can mimic model degradation.

Checkpoint: Metrics show request volume, latency, and failure trend.

3. Execute the operational step.
   az monitor metrics alert create --name model-drift-alert --resource-group rg-ai300 --scopes
   Command type: Azure Monitor CLI verification; confirm metric names for the selected Azure resource type.

Reason: Create an alert so drift or quality thresholds produce an operational action instead of passive dashboard data.

Checkpoint: Alert rule is enabled and bound to an action group.

4. Execute the operational step.
   az ml job create --file retrain-pipeline.yml -g rg-ai300 -w mlw-ai300-prod
   Command type: Azure ML CLI verification; confirm extension/version in the lab environment.

Reason: Trigger retraining through a pipeline so new data, evaluation, registration, and promotion remain auditable.

Checkpoint: Pipeline completes and produces a candidate model with evaluation metrics.

Technical Chain

A user, workflow, or deployment command targets Data drift monitor and submits configuration to Azure control plane or a project runtime. Azure validates identity, resource scope, quota, version references, and network reachability because the runtime cannot safely use an object that is not authorized, versioned, reachable, or measurable. The configured object then participates in the runtime path through Model metric, Alert rule, Retraining pipeline. This sequence works because each object unlocks the next dependency: identity allows access, versioning allows reproducibility, network resolution allows execution, and telemetry allows verification. When the workload executes, telemetry, status output, logs, API response, or evaluation metrics prove whether the chain is complete. If the chain breaks, the failure appears as the operational symptom described in the scenario: Bad predictions persist when production telemetry is collected but no alert, threshold, or retraining action is bound to it. An incorrect configuration creates the observed failure because it changes a nearby object while leaving the actual missing dependency unresolved.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

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

| Inspect deployment logs | `az ml online-deployment get-logs --endpoint-name churn-endpoint --name blue -g rg-ai300 -w mlw-ai300-prod` | Logs distinguish scoring failure from quality degradation. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

| Review endpoint metrics | `az monitor metrics list --resource <endpoint-resource-id> --metric Requests,Latency` | Request and latency trends are visible. Command type: Azure Monitor CLI verification; confirm metric names for the selected Azure resource type. |

| Verify alert rule | `az monitor metrics alert show --name model-drift-alert --resource-group rg-ai300` | Alert rule is enabled and scoped to the monitor resource. Command type: Azure Monitor CLI verification; confirm metric names for the selected Azure resource type. |

| Check retraining pipeline | `az ml job show --name <retrain-job> -g rg-ai300 -w mlw-ai300-prod --query status` | Retraining pipeline completes and emits candidate model outputs. Command type: Azure ML CLI verification; confirm extension/version in the lab environment. |

Practice Questions

1. A training pipeline must run the same preprocessing, training, and evaluation steps whenever new data is approved. Which Azure Machine Learning feature best supports this workflow?
   A. Manual notebook execution only
   B. Azure Machine Learning pipeline jobs with reusable components
   C. A public storage container with anonymous read access
   D. A single unmanaged virtual machine

2. A model has been trained successfully, and the team needs to promote it only if evaluation metrics meet release criteria. What should happen before production deployment?
   A. Disable Application Insights for the endpoint
   B. Delete all previous model versions to avoid confusion
   C. Increase compute cluster maximum nodes
   D. Register the model version with metrics and compare it against acceptance thresholds

3. A production online endpoint serves a new model version to 10 percent of traffic while the previous model keeps 90 percent. What operation is being used?
   A. Batch inference
   B. Data asset versioning
   C. Traffic splitting between deployments
   D. Workspace private endpoint creation

4. A batch scoring job must process a large file set every night and write predictions to storage. Which deployment pattern fits best?
   A. Managed batch endpoint or scheduled batch pipeline
   B. Real-time online endpoint with one request per file from a browser
   C. Prompt template A/B testing only
   D. Manual scoring on a laptop

5. A deployed model starts returning higher latency after a new version is released. Which telemetry should be reviewed first?
   A. The Key Vault soft-delete retention period only
   B. The workspace display name
   C. The number of files in the source repository root
   D. Endpoint request latency, error rate, instance utilization, and deployment logs

6. A team needs to roll back from model version 5 to version 4 after a regression is detected. What prior practice makes this rollback practical?
   A. Storing only the latest model file with the same name
   B. Keeping versioned registered models and separate endpoint deployments
   C. Removing MLflow tracking data after each run
   D. Using anonymous access for the datastore

7. A data scientist asks why MLflow tracking is required in the training pipeline. What is the best operational reason?
   A. It automatically creates private endpoints for every service
   B. It replaces all Azure RBAC assignments
   C. It records parameters, metrics, artifacts, and run lineage needed for comparison and audit
   D. It tunes all prompts used by generative AI applications

8. A model's accuracy drops in production because input data distribution has shifted from the training baseline. Which operational capability should detect this condition?

A. Data drift or model monitoring configured with production data and baseline statistics
B. A workspace name change
C. A larger Key Vault SKU
D. A source-control branch rename

9. A release pipeline deploys a model, but health probes fail immediately. What should the engineer inspect first?
A. The vector index HNSW parameter only
B. The number of users assigned Reader access to the subscription
C. The prompt safety policy in a Foundry project
D. Scoring script initialization, environment dependencies, container logs, and endpoint deployment state

10. A team wants release automation to stop if a candidate model performs worse than the current production model. Where should this gate be placed?
A. After deleting the existing endpoint
B. In the CI/CD workflow after evaluation and before production traffic is shifted
C. Only after all production users report issues
D. In the storage account firewall rule description

Design and implement a GenAIOps infrastructure

Implement Foundry environments and platform configuration

Exam Radar

- Official Blueprint Mapping: Create and configure Foundry resources and project environments; Configure identity and access management with managed identities and role-based access control (RBAC); Implement network security and private networking configurations; Deploy infrastructure using Bicep templates and Azure CLI.
- Domain Weight: Design and implement a GenAIOps infrastructure represents 20-25% of the official skills measured.
- Core Priority: The exam tests whether the candidate can operate this sub-skill as a concrete Azure workflow inside Design and implement a GenAIOps infrastructure, not merely identify the service name.
- High Frequency: Expect scenario wording that combines Microsoft Foundry resource, Microsoft Foundry project, Managed identity, and verification evidence from commands, logs, metrics, or portal state.
- Confusion Alert: A valid token is insufficient when the client resolves the public endpoint while public network access is disabled.

- Scenario Logic: Choose the answer that creates a verifiable dependency chain: configure the object, bind identity or version, run the operation, then prove the resulting state.
- Version Delta: AI-300 combines Azure Machine Learning and Microsoft Foundry under AIOps, so this point must be read as an operational platform task rather than a standalone concept.
- Failure Trigger: GenAI applications fail before inference when project RBAC, managed identity binding, private endpoint approval, or DNS integration is incomplete.
- Operational Dependency: The task depends on Microsoft Foundry resource, Microsoft Foundry project, Managed identity, RBAC assignment, Private endpoint, Bicep deployment and a validation step that proves the configured state is actually usable.
- How the Exam Asks It: A GenAI application cannot call a Microsoft Foundry model deployment from a private network even though the application has a valid identity token.
- How Distractors Are Designed: Distractors rotate keys, change the prompt, select another model, or add application logging. These do not fix RBAC scope or private endpoint name resolution.
- Why the Correct Answer Works: The correct answer configures project/resource access, managed identity permissions, private endpoint approval, and DNS resolution because inference requires both authorization and reachable network path.

Atomic Deconstruction - Operational Level

Microscopic technical focus: Configuring Microsoft Foundry resources, projects, managed identities, RBAC, private networking, Bicep, and Azure CLI deployment.

Beginner explanation: Microsoft Foundry configuration gives GenAI apps a governed project, identity boundary, and network path. A model call must pass both permission and connectivity checks.

Operational split for this point: start with Microsoft Foundry resource, then verify Microsoft Foundry project and Managed identity before trusting any production outcome. The exam is testing whether the candidate can locate the missing dependency, not whether the candidate recognizes every service name in the scenario.

For this knowledge point, the target objects are Microsoft Foundry resource, Microsoft Foundry project, Managed identity, RBAC assignment, Private endpoint, Bicep deployment. The exam usually describes one broken link in that chain. The correct answer is the option that restores the missing operational dependency rather than the option that only describes the platform at a high level.

Why-layer: Microsoft Foundry resource becomes exam-relevant only when the surrounding dependency chain can run. In this topic, GenAI applications fail before inference when project RBAC, managed identity binding, private endpoint approval, or DNS integration is incomplete. The correct configuration matters because it changes the state that controls execution, authorization, resolution, evaluation, or observability; a nearby but unrelated action leaves the same failure mode in place.

Decision tree: if the scenario describes access failure, inspect identity and RBAC before changing compute or code; if it describes unresolved assets, inspect name, version, and scope; if it describes runtime failure,

inspect logs, endpoint invocation, metrics, or evaluation output; if it describes quality degradation, inspect data, retrieval, evaluation, and monitoring evidence before changing the model.

Common mistakes: Selecting a familiar Azure service without checking the missing dependency in the scenario. Treating a successful create operation as proof of runtime behavior. Choosing a monitoring action when the scenario asks for configuration or access remediation.

Practice question: A GenAI application cannot call a Microsoft Foundry model deployment from a private network even though the application has a valid identity token.
A. Assign the application identity the required Microsoft Foundry or Azure OpenAI resource role and validate private endpoint DNS resolution from the application network.
B. Rotate the API key because every Microsoft Foundry access issue is a credential leak.
C. Switch to a different foundation model deployment without changing network or RBAC settings.
D. Add more application logging before testing private endpoint connectivity.

expected answer: A

Explanation: A is correct because model invocation requires both authorization and network reachability. B, C, and D do not repair the missing RBAC or private DNS dependency.

The common decision point is: A valid token is insufficient when the client resolves the public endpoint while public network access is disabled. Therefore, read every scenario for the actor, the resource scope, the object version, the network path, the metric threshold, and the expected observable result.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Microsoft Foundry project | Access boundary | User, group, service principal, managed identity | No project operation until role assigned | Microsoft Foundry resource and Entra ID | Deployment or evaluation action is denied |

| Managed identity | Token use | System-assigned or user-assigned | Not bound to caller by default | Application hosting service | API request receives 401 or 403 |
| Private endpoint | Connection state | Pending, approved, rejected | No private route until approved | VNet and resource networking | Client cannot reach endpoint |

| Private DNS | Name resolution | Private IP for service FQDN | Public resolution by default | DNS zone link | Requests route to blocked public endpoint |

Step-by-Step Execution Path

1. Execute the operational step.
   az role assignment create --assignee --role "Cognitive Services User" --scope

Command type: Azure CLI RBAC verification for Entra identity and Azure AI resource scope.

Reason: Assign the caller identity at the resource scope because token possession alone does not grant model invocation rights.

Checkpoint: Role assignment list shows Cognitive Services User for the managed identity.

2. Execute the operational step.
   az network private-endpoint-connection list --id
   Command type: Azure CLI network verification for the Microsoft Foundry or Azure OpenAI resource; confirm the exact resource ID from the active environment.

Reason: Check private endpoint approval because a pending connection does not create a usable private route; confirm the exact resource ID from Microsoft Foundry management center or Azure resource properties.

Checkpoint: Connection state is Approved.

3. Execute the operational step.
   nslookup .openai.azure.com
   Command type: network/DNS rehearsal command for private endpoint validation.

Reason: Validate DNS from the application network because private endpoint traffic depends on resolving the public FQDN to a private IP.

Checkpoint: Name resolves to a private address.

4. Execute the operational step.
   curl -H "Authorization: Bearer " https://.openai.azure.com/openai/deployments//chat/completions?api-version=
   Command type: network/API rehearsal command for the selected app or Azure OpenAI endpoint; confirm URL, header, and API version before use.

Reason: Call the endpoint after RBAC and DNS checks to prove inference works from the intended network.

Checkpoint: API returns a model response rather than 401, 403, or network timeout.

Technical Chain

A user, workflow, or deployment command targets Microsoft Foundry resource and submits configuration to Azure control plane or a project runtime. Azure validates identity, resource scope, quota, version references, and network reachability because the runtime cannot safely use an object that is not authorized, versioned, reachable, or measurable. The configured object then participates in the runtime path through Microsoft Foundry project, Managed identity, RBAC assignment. This sequence works because each object unlocks the next dependency: identity allows access, versioning allows reproducibility, network resolution allows execution, and telemetry allows verification. When the workload executes, telemetry, status output, logs, API response, or evaluation metrics prove whether the chain is complete. If the chain breaks, the failure appears

as the operational symptom described in the scenario: GenAI applications fail before inference when project RBAC, managed identity binding, private endpoint approval, or DNS integration is incomplete. An incorrect configuration creates the observed failure because it changes a nearby object while leaving the actual missing dependency unresolved.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

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

| Verify caller authorization | `az role assignment list --assignee <principal-id> --scope <foundry-resource-id> -o table` | The application identity has the required Microsoft Foundry or Azure OpenAI resource role at the expected scope. Command type: Azure CLI RBAC verification for Entra identity and Azure AI resource scope. |

| Check private endpoint approval | `az network private-endpoint-connection list --id <foundry-resource-id>` | The private endpoint connection is Approved before private traffic is expected to work. Command type: Azure CLI network verification for the Microsoft Foundry or Azure OpenAI resource; confirm the exact resource ID from the active environment. |

| Validate private DNS from workload network | `nslookup <resource-name>.openai.azure.com` | The service FQDN resolves to a private address from the application network. Command type: network/DNS rehearsal command for private endpoint validation. |

| Prove endpoint reachability | `curl -H "Authorization: Bearer <token>" https://<resource-name>.openai.azure.com/openai/deployments/<deployment>/chat/completions?api-version=<version>` | The call returns a model response instead of authorization or network errors. Command type: network/API rehearsal command for the selected app or Azure OpenAI endpoint; confirm URL, header, and API version before use. |

Deploy and manage foundation models for production workloads

Exam Radar

- Official Blueprint Mapping: Deploy foundation models by using serverless API endpoints and managed compute options; Select appropriate models for specific use cases; Implement model versioning and production deployment strategies; Configure provisioned throughput units for high-volume workloads.
- Domain Weight: Design and implement a GenAIOps infrastructure represents 20-25% of the official skills measured.

- Core Priority: The exam tests whether the candidate can operate this sub-skill as a concrete Azure workflow inside Design and implement a GenAIOps infrastructure, not merely identify the service name.
- High Frequency: Expect scenario wording that combines Foundation model, Model deployment, Serverless API endpoint, and verification evidence from commands, logs, metrics, or portal state.
- Confusion Alert: Model selection must balance latency, context window, modality, region, cost, and throughput instead of choosing the largest model by default.
- Scenario Logic: Choose the answer that creates a verifiable dependency chain: configure the object, bind identity or version, run the operation, then prove the resulting state.
- Version Delta: AI-300 combines Azure Machine Learning and Microsoft Foundry under AIOps, so this point must be read as an operational platform task rather than a standalone concept.
- Failure Trigger: High-volume workloads receive throttling or unstable latency when provisioned throughput and quota are not planned.
- Operational Dependency: The task depends on Foundation model, Model deployment, Serverless API endpoint, Managed compute option, Provisioned throughput unit, Model version and a validation step that proves the configured state is actually usable.
- How the Exam Asks It: A workload needs predictable latency and capacity for a selected foundation model, and the team must decide between available deployment and throughput options.
- How Distractors Are Designed: Distractors choose the largest model, ignore region availability, increase prompt length, or monitor only total calls. These do not reserve capacity or match model constraints.
- Why the Correct Answer Works: The correct answer selects a supported model/version/region and configures the appropriate deployment capacity or provisioned throughput based on latency, volume, quota, and cost constraints.

Atomic Deconstruction - Operational Level

Microscopic technical focus: Selecting models, deploying serverless endpoints, managing versions, and configuring provisioned throughput.

Beginner explanation: A foundation model deployment is not just model selection. It is a capacity, version, region, quota, and endpoint decision.

Operational split for this point: start with Foundation model, then verify Model deployment and Serverless API endpoint before trusting any production outcome. The exam is testing whether the candidate can locate the missing dependency, not whether the candidate recognizes every service name in the scenario.

For this knowledge point, the target objects are Foundation model, Model deployment, Serverless API endpoint, Managed compute option, Provisioned throughput unit, Model version. The exam usually describes one broken link in that chain. The correct answer is the option that restores the missing operational dependency rather than the option that only describes the platform at a high level.

Why-layer: Foundation model becomes exam-relevant only when the surrounding dependency chain can run. In this topic, High-volume workloads receive throttling or unstable latency when provisioned throughput and quota are not planned. The correct configuration matters because it changes the state that controls execution, authorization, resolution, evaluation, or observability; a nearby but unrelated action leaves the same failure mode in place.

Decision tree: if the scenario describes access failure, inspect identity and RBAC before changing compute or code; if it describes unresolved assets, inspect name, version, and scope; if it describes runtime failure, inspect logs, endpoint invocation, metrics, or evaluation output; if it describes quality degradation, inspect data, retrieval, evaluation, and monitoring evidence before changing the model.

Common mistakes: Selecting a familiar Azure service without checking the missing dependency in the scenario. Treating a successful create operation as proof of runtime behavior. Choosing a monitoring action when the scenario asks for configuration or access remediation.

Practice question: A workload needs predictable latency and capacity for a selected foundation model, and the team must decide between available deployment and throughput options.
A. Select a supported model/version/region and configure deployment capacity or provisioned throughput based on latency and volume requirements.
B. Choose the largest available model because larger models always reduce production latency.
C. Increase max output tokens to prevent deployment throttling.
D. Monitor total calls only after users begin receiving 429 responses.

expected answer: A

Explanation: A is correct because production deployment is a model, region, quota, and capacity decision. B can increase latency/cost, C affects generation length, and D detects throttling after capacity planning failed.

The common decision point is: Model selection must balance latency, context window, modality, region, cost, and throughput instead of choosing the largest model by default. Therefore, read every scenario for the actor, the resource scope, the object version, the network path, the metric threshold, and the expected observable result.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Model deployment | Model and version | Supported model catalog entry and version | No endpoint until deployed | Region availability and quota | Application references deployment that does not exist |

| Serverless endpoint | Capacity mode | Provider-managed throughput | No reserved isolation | Model support and endpoint access | Variable latency under load |
| Provisioned throughput | Reserved capacity | Throughput unit count | Unavailable until quota approved | Supported model and region | 429 throttling or missed latency target |

| Deployment metric | Operational signal | Total calls, latency, throttling, token usage | Unobserved until monitored | Azure Monitor metric stream | Capacity issue is misdiagnosed |

Step-by-Step Execution Path

1. Execute the operational step.
   az cognitiveservices account deployment list -g rg-ai300-genai -n
   Command type: Azure CLI verification for Azure OpenAI/Cognitive Services deployment state; confirm current parameters, region support, and model availability.

Reason: List deployments first so the application uses a deployment name that actually exists in the target account.

Checkpoint: The intended deployment appears with succeeded state.

2. Execute the operational step.
   az cognitiveservices account show-usage -g rg-ai300-genai -n
   Command type: Azure CLI verification for Azure OpenAI/Cognitive Services deployment state; confirm current parameters, region support, and model availability.

Reason: Check quota before deployment because capacity failures are quota constraints, not prompt or application bugs.

Checkpoint: Usage output shows available quota for the selected model family.

3. Execute the operational step.
   az cognitiveservices account deployment create --name --resource-group rg-ai300-genai --deployment-name gpt-prod --model-name --model-version --model-format OpenAI --sku-name Standard --sku-capacity 30
   Command type: Azure CLI verification for Azure OpenAI/Cognitive Services deployment state; confirm current parameters, region support, and model availability.

Reason: Create the deployment with explicit model and capacity so runtime calls target a stable endpoint configuration.

Checkpoint: Deployment provisioning state is Succeeded.

4. Execute the operational step.
   az monitor metrics list --resource --metric TotalCalls,ThrottledCalls,Latency
   Command type: Azure Monitor CLI verification; confirm metric names for the selected Azure resource type.

Reason: Monitor deployment metrics because successful creation does not prove capacity is adequate under production traffic.

Checkpoint: Metrics show call volume, throttling, and latency by time window.

Technical Chain

A user, workflow, or deployment command targets Foundation model and submits configuration to Azure control plane or a project runtime. Azure validates identity, resource scope, quota, version references, and network reachability because the runtime cannot safely use an object that is not authorized, versioned, reachable, or measurable. The configured object then participates in the runtime path through Model deployment, Serverless API endpoint, Managed compute option. This sequence works because each object unlocks the next dependency: identity allows access, versioning allows reproducibility, network resolution allows execution, and telemetry allows verification. When the workload executes, telemetry, status output, logs, API response, or evaluation metrics prove whether the chain is complete. If the chain breaks, the failure appears as the operational symptom described in the scenario: High-volume workloads receive throttling or unstable latency when provisioned throughput and quota are not planned. An incorrect configuration creates the observed failure because it changes a nearby object while leaving the actual missing dependency unresolved.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| List model deployments | `az cognitiveservices account deployment list -g rg-ai300-genai -n <account>` | The intended deployment name appears in the target Azure OpenAI account. Command type: Azure CLI verification for Azure OpenAI/Cognitive Services deployment state; confirm current parameters, region support, and model availability. |

| Inspect account quota | `az cognitiveservices account show-usage -g rg-ai300-genai -n <account>` | Usage and limit values show whether requested capacity is available. Command type: Azure CLI verification for Azure OpenAI/Cognitive Services deployment state; confirm current parameters, region support, and model availability. |
| Verify deployment state | `az cognitiveservices account deployment show --name <account> --resource-group rg-ai300-genai --deployment-name gpt-prod` | Provisioning state and model/version match the release plan. Command type: Azure CLI verification for Azure OpenAI/Cognitive Services deployment state; confirm current parameters, region support, and model availability. |

| Review serving metrics | `az monitor metrics list --resource <deployment-resource-id> --metric TotalCalls,ThrottledCalls,Latency` | Metrics expose traffic, throttling, and latency after deployment. Command type: Azure Monitor CLI verification; confirm metric names for the selected Azure resource type. |

Implement prompt versioning and management with source control

Exam Radar

- Official Blueprint Mapping: Design and develop prompts; Create prompt variants and compare performance across different prompts; Implement version control for prompts by using Git repositories.
- Domain Weight: Design and implement a GenAIOps infrastructure represents 20-25% of the official skills measured.
- Core Priority: The exam tests whether the candidate can operate this sub-skill as a concrete Azure workflow inside Design and implement a GenAIOps infrastructure, not merely identify the service name.
- High Frequency: Expect scenario wording that combines Prompt file, Prompt variant, Evaluation dataset, and verification evidence from commands, logs, metrics, or portal state.
- Confusion Alert: Editing a production prompt in a UI gives speed but loses repeatable comparison, review, rollback, and release traceability.
- Scenario Logic: Choose the answer that creates a verifiable dependency chain: configure the object, bind identity or version, run the operation, then prove the resulting state.
- Version Delta: AI-300 combines Azure Machine Learning and Microsoft Foundry under AIOps, so this point must be read as an operational platform task rather than a standalone concept.
- Failure Trigger: A prompt regression cannot be diagnosed when the active prompt text, model deployment, dataset, and metric output are not tied to a commit.
- Operational Dependency: The task depends on Prompt file, Prompt variant, Evaluation dataset, Git branch, Pull request, Release tag and a validation step that proves the configured state is actually usable.
- How the Exam Asks It: A prompt change improves one test case but breaks production behavior, and the team needs review, evaluation evidence, and rollback.
- How Distractors Are Designed: Distractors edit the prompt directly in a portal, rename the prompt file, or rely on chat history screenshots. These do not provide release traceability.
- Why the Correct Answer Works: The correct answer stores prompt variants, evaluation datasets, and metric outputs in source control with pull-request review because prompt changes are deployable artifacts.

Atomic Deconstruction - Operational Level

Microscopic technical focus: Designing prompt variants, comparing prompt performance, and controlling releases through Git repositories.

Beginner explanation: A prompt is production logic. Treat prompt text like code: version it, review it, evaluate it, and keep a rollback point.

Operational split for this point: start with Prompt file, then verify Prompt variant and Evaluation dataset before trusting any production outcome. The exam is testing whether the candidate can locate the missing dependency, not whether the candidate recognizes every service name in the scenario.

For this knowledge point, the target objects are Prompt file, Prompt variant, Evaluation dataset, Git branch, Pull request, Release tag. The exam usually describes one broken link in that chain. The correct answer is the option that restores the missing operational dependency rather than the option that only describes the platform at a high level.

Why-layer: Prompt file becomes exam-relevant only when the surrounding dependency chain can run. In this topic, A prompt regression cannot be diagnosed when the active prompt text, model deployment, dataset, and metric output are not tied to a commit. The correct configuration matters because it changes the state that controls execution, authorization, resolution, evaluation, or observability; a nearby but unrelated action leaves the same failure mode in place.

Decision tree: if the scenario describes access failure, inspect identity and RBAC before changing compute or code; if it describes unresolved assets, inspect name, version, and scope; if it describes runtime failure, inspect logs, endpoint invocation, metrics, or evaluation output; if it describes quality degradation, inspect data, retrieval, evaluation, and monitoring evidence before changing the model.

Common mistakes: Selecting a familiar Azure service without checking the missing dependency in the scenario. Treating a successful create operation as proof of runtime behavior. Choosing a monitoring action when the scenario asks for configuration or access remediation.

Practice question: A prompt change improves one test case but breaks production behavior, and the team needs review, evaluation evidence, and rollback.
A. Commit the prompt variant and evaluation dataset to Git, run evaluation in CI, and tag the approved prompt version.
B. Edit the production prompt directly in the portal and document the change in chat history.
C. Rename the prompt file so reviewers can distinguish it from the old version.
D. Rely on a few manual chat transcripts as release evidence.

expected answer: A

Explanation: A is correct because prompt changes need review, evaluation evidence, and rollback. B bypasses source control, C is naming not governance, and D is anecdotal testing.

The common decision point is: Editing a production prompt in a UI gives speed but loses repeatable comparison, review, rollback, and release traceability. Therefore, read every scenario for the actor, the resource scope, the object version, the network path, the metric threshold, and the expected observable result.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ----------
---------- |
| Prompt file | Version source | Git commit, branch, tag | Uncontrolled text until committed | Repository and review process | Active prompt cannot be traced |

| Evaluation dataset | Regression cases | JSONL or tabular test cases | Not linked until committed | Prompt evaluation workflow | Prompt passes anecdotal testing only |
| Pull request | Review gate | Approvals, checks, comments | No gate on direct edit | Branch protection and CI | Regression enters production |

| Release tag | Rollback pointer | Semantic tag or commit SHA | No rollback marker | Deployment process | Previous stable prompt cannot be restored |

Step-by-Step Execution Path

1. Execute the operational step.
   git checkout -b prompt/claims-routing-v2
   Command type: Git source-control verification.

Reason: Use a branch so prompt experimentation is isolated from the production prompt path.

Checkpoint: Branch name appears in git status.

2. Execute the operational step.
   git add prompts/claims-routing.prompt.yml evaluations/claims-routing.dataset.jsonl
   Command type: Git source-control verification.

Reason: Commit prompt and evaluation data together because a prompt version without its test evidence is not release-ready.

Checkpoint: Git status shows both files staged.

3. Execute the operational step.
   gh workflow run evaluate-prompts.yml --ref prompt/claims-routing-v2
   Command type: GitHub CLI workflow verification.

Reason: Run evaluation before merge so quality and safety regressions block the prompt release.

Checkpoint: Workflow result contains metric output for the branch.

4. Execute the operational step.
   git tag prompt-claims-routing-v2-approved
   Command type: Git source-control verification.

Reason: Tag the approved prompt commit so rollback can target a known stable version.

Checkpoint: Git log and tag list show the approved release point.

Technical Chain

A user, workflow, or deployment command targets Prompt file and submits configuration to Azure control plane or a project runtime. Azure validates identity, resource scope, quota, version references, and network reachability because the runtime cannot safely use an object that is not authorized, versioned, reachable, or measurable. The configured object then participates in the runtime path through Prompt variant, Evaluation dataset, Git branch. This sequence works because each object unlocks the next dependency: identity allows access, versioning allows reproducibility, network resolution allows execution, and telemetry allows verification. When the workload executes, telemetry, status output, logs, API response, or evaluation metrics prove whether the chain is complete. If the chain breaks, the failure appears as the operational symptom described in the scenario: A prompt regression cannot be diagnosed when the active prompt text, model deployment, dataset, and metric output are not tied to a commit. An incorrect configuration creates the observed failure because it changes a nearby object while leaving the actual missing dependency unresolved.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

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

| Check active branch | `git status --short --branch` | Prompt work is isolated on the expected feature branch. Command type: Git source-control verification. |

| Verify prompt and evaluation files | `git diff --name-only --cached` | Prompt and evaluation dataset changes are staged together. Command type: Git source-control verification. |

| Inspect evaluation workflow result | `gh run view <run-id> --log` | Evaluation workflow logs include metric output for the prompt change. Command type: GitHub CLI workflow verification. |

| Confirm release tag | `git tag --list prompt-claims-routing-v2-approved` | The approved prompt version has a rollback marker. Command type: Git source-control verification. |

Practice Questions

1. A team is configuring Microsoft Foundry for multiple application teams. They need project-level separation, managed identity access, and governed model deployment. Which design best fits?
   A. Use shared personal accounts for every project
   B. Store all prompts in a local text file with no version history
   C. Configure Foundry hubs/projects with RBAC, managed identities, and controlled deployment permissions
   D. Disable network controls to simplify access

2. A generative AI app receives 403 errors when calling a deployed foundation model from a managed identity. What should be verified first?
   A. The batch endpoint output folder

B. The number of chunks in the retrieval index
C. The markdown formatting of the system prompt
D. The managed identity's role assignment and target resource scope

3. A production app must avoid surprise downtime when switching to a new foundation model deployment. Which practice should be used?
   A. Deploy the new model separately, validate it, then update traffic or application configuration through a controlled release
   B. Replace the model name directly in production code with no validation
   C. Delete telemetry before the release to reduce noise
   D. Remove all content filters temporarily

4. A prompt change fixes one scenario but degrades another. What should the team implement to control this lifecycle?
   A. Manual edits in the production portal only
   B. Prompt versioning in source control with evaluation records for each candidate prompt
   C. Larger GPU compute for all training jobs
   D. Anonymous public access to prompt files

5. A model deployment is throttling during business hours. Which capacity signal should be reviewed?
   A. MLflow run tags from a classical training job
   B. Storage account blob soft-delete settings only
   C. Provisioned throughput, quota limits, token usage, and request rate metrics
   D. The number of pull requests in the repository

6. A security review requires that app secrets not be embedded in prompt orchestration code. What should the engineer use?
   A. Prompt examples that include the secret value
   B. Hard-coded API keys in environment-specific branches
   C. Plain text variables committed to the repository
   D. Managed identity and Key Vault or platform-managed secret references

7. A Foundry project is accessible from a developer laptop but not from the production application subnet after private networking is enabled. What is the most relevant check?
   A. Private endpoint DNS resolution, network rules, and subnet routing for the production path
   B. Whether the model card includes a business description
   C. Whether a training dataset has balanced classes
   D. Whether the prompt contains fewer than five examples

8. A GitHub Actions workflow updates prompt templates and deployment configuration. The team wants deployment only after peer review. What should be required?
   A. Direct pushes to the production branch by any developer
   B. Pull request review and environment protection before applying changes

C. Manual copy-and-paste from a local notebook
D. Disabling branch history after each release

9. A generative AI app must use a specific model deployment name across environments while allowing different capacities per environment. How should configuration be handled?
   A. Use a random deployment name each time
   B. Hard-code all values in the application binary
   C. Use parameterized deployment configuration with environment-specific values
   D. Store configuration only in a chat transcript
10. A release fails because the application calls an old prompt version even after a repository update. Which evidence should be checked?
    A. The model registry description field
    B. The size of the Azure Machine Learning compute cluster
    C. The storage account replication setting only
    D. Build artifact contents, deployment logs, and runtime configuration that selects the prompt version

Implement generative AI quality assurance and observability

Configure evaluation and validation for generative AI applications and agents

Exam Radar

- Official Blueprint Mapping: Create test datasets and data mapping for comprehensive model evaluation; Implement AI quality metrics, including groundedness, relevance, coherence, and fluency; Configure risk and safety evaluations for harmful content detection; Set up automated evaluation workflows by using built-in and custom evaluation metrics.
- Domain Weight: Implement generative AI quality assurance and observability represents 10-15% of the official skills measured.
- Core Priority: The exam tests whether the candidate can operate this sub-skill as a concrete Azure workflow inside Implement generative AI quality assurance and observability, not merely identify the service name.
- High Frequency: Expect scenario wording that combines Evaluation dataset, Data mapping, Groundedness metric, and verification evidence from commands, logs, metrics, or portal state.
- Confusion Alert: Manual answer review cannot replace repeatable evaluation when prompt and model versions must be compared before release.
- Scenario Logic: Choose the answer that creates a verifiable dependency chain: configure the object, bind identity or version, run the operation, then prove the resulting state.

- Version Delta: AI-300 combines Azure Machine Learning and Microsoft Foundry under AIOps, so this point must be read as an operational platform task rather than a standalone concept.
- Failure Trigger: Scores are misleading when question, context, expected answer, and labels are mapped to the wrong evaluator fields.
- Operational Dependency: The task depends on Evaluation dataset, Data mapping, Groundedness metric, Safety evaluator, Custom metric, Evaluation workflow and a validation step that proves the configured state is actually usable.
- How the Exam Asks It: A GenAI app returns plausible but unsupported answers, and the team needs a repeatable evaluation gate before release.
- How Distractors Are Designed: Distractors only inspect a few manual chat transcripts, raise temperature, or add more logs. These do not measure groundedness, relevance, or safety against a dataset.
- Why the Correct Answer Works: The correct answer maps evaluation dataset fields to built-in or custom evaluators and blocks release when groundedness, relevance, safety, or custom metrics fail thresholds.

Atomic Deconstruction - Operational Level

Microscopic technical focus: Mapping test datasets to groundedness, relevance, coherence, fluency, safety, built-in metrics, and custom metrics.

Beginner explanation: Evaluation converts subjective response quality into repeatable checks. Dataset mapping is the first dependency because metrics are meaningless if fields are wrong.

Operational split for this point: start with Evaluation dataset, then verify Data mapping and Groundedness metric before trusting any production outcome. The exam is testing whether the candidate can locate the missing dependency, not whether the candidate recognizes every service name in the scenario.

For this knowledge point, the target objects are Evaluation dataset, Data mapping, Groundedness metric, Safety evaluator, Custom metric, Evaluation workflow. The exam usually describes one broken link in that chain. The correct answer is the option that restores the missing operational dependency rather than the option that only describes the platform at a high level.

Why-layer: Evaluation dataset becomes exam-relevant only when the surrounding dependency chain can run. In this topic, Scores are misleading when question, context, expected answer, and labels are mapped to the wrong evaluator fields. The correct configuration matters because it changes the state that controls execution, authorization, resolution, evaluation, or observability; a nearby but unrelated action leaves the same failure mode in place.

Decision tree: if the scenario describes access failure, inspect identity and RBAC before changing compute or code; if it describes unresolved assets, inspect name, version, and scope; if it describes runtime failure, inspect logs, endpoint invocation, metrics, or evaluation output; if it describes quality degradation, inspect data, retrieval, evaluation, and monitoring evidence before changing the model.

Common mistakes: Selecting a familiar Azure service without checking the missing dependency in the scenario. Treating a successful create operation as proof of runtime behavior. Choosing a monitoring action when the scenario asks for configuration or access remediation.

Practice question: A GenAI app returns plausible but unsupported answers, and the team needs a repeatable evaluation gate before release.
A. Map question, context, expected answer, and safety labels correctly, then run quality and safety evaluators with release thresholds.
B. Review five generated responses manually and approve the prompt if they look reasonable.
C. Increase model temperature to improve groundedness scores.
D. Add more application logs instead of creating an evaluation dataset.

expected answer: A

Explanation: A is correct because evaluation depends on mapped test data and measurable gates. B is not repeatable, C can worsen consistency, and D observes requests but does not score output quality.

The common decision point is: Manual answer review cannot replace repeatable evaluation when prompt and model versions must be compared before release. Therefore, read every scenario for the actor, the resource scope, the object version, the network path, the metric threshold, and the expected observable result.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Evaluation dataset | Field mapping | Input, context, expected answer, labels | Rows are inert until mapped | Evaluator schema | Scores are invalid or missing |

| Groundedness metric | Context support | Score or pass/fail threshold | No release gate | Retrieved context or reference data | Unsupported answers pass review |
| Safety evaluator | Risk category | Hate, violence, self-harm, sexual, custom policy | Not enforced until configured | Safety test set | Unsafe output reaches production |

| Custom metric | Domain-specific rule | Function, rubric, or evaluator prompt | No domain gate | Business acceptance criteria | Correct-looking answer violates domain rule |

Step-by-Step Execution Path

1. Execute the operational step.
   python evaluate.py --dataset evaluations/app.jsonl --dry-run
   Command type: local lab rehearsal script, not an official Azure command.

Reason: Dry-run mapping first because a mislabeled context or expected-answer field makes every score misleading.

Checkpoint: Output shows required evaluator fields mapped.

2. Execute the operational step.
   python evaluate.py --dataset evaluations/app.jsonl --metrics groundedness relevance coherence fluency
   Command type: local lab rehearsal script, not an official Azure command.

Reason: Run quality metrics so the release decision is based on repeatable scores rather than manual impressions.

Checkpoint: Aggregate metric report is produced for the prompt or app version.

3. Execute the operational step.
   python evaluate.py --dataset evaluations/safety.jsonl --metrics safety
   Command type: local lab rehearsal script, not an official Azure command.

Reason: Run safety cases separately because functional correctness does not prove safe behavior under adversarial prompts.

Checkpoint: Unsafe cases are flagged and threshold failures return nonzero status.

4. Execute the operational step.
   gh workflow run evaluate-prompts.yml
   Command type: GitHub CLI workflow verification.

Reason: Execute evaluation in CI so release approval depends on the same gate every time.

Checkpoint: Workflow summary stores metric artifacts and pass/fail status.

Technical Chain

A user, workflow, or deployment command targets Evaluation dataset and submits configuration to Azure control plane or a project runtime. Azure validates identity, resource scope, quota, version references, and network reachability because the runtime cannot safely use an object that is not authorized, versioned, reachable, or measurable. The configured object then participates in the runtime path through Data mapping, Groundedness metric, Safety evaluator. This sequence works because each object unlocks the next dependency: identity allows access, versioning allows reproducibility, network resolution allows execution, and telemetry allows verification. When the workload executes, telemetry, status output, logs, API response, or evaluation metrics prove whether the chain is complete. If the chain breaks, the failure appears as the operational symptom described in the scenario: Scores are misleading when question, context, expected answer, and labels are mapped to the wrong evaluator fields. An incorrect configuration creates the observed failure because it changes a nearby object while leaving the actual missing dependency unresolved.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| --- | --- | --- |
| Dry-run evaluation mapping | `python evaluate.py --dataset evaluations/app.jsonl --dry-run` | Required fields map to evaluator inputs before scoring. Command type: local lab rehearsal script, not an official Azure command. |

| Review quality metrics | `python evaluate.py --dataset evaluations/app.jsonl --metrics groundedness relevance coherence fluency` | Metric output is produced for the prompt or app version. Command type: local lab rehearsal script, not an official Azure command. |
| Review safety metrics | `python evaluate.py --dataset evaluations/safety.jsonl --metrics safety` | Unsafe cases are flagged and threshold failures are visible. Command type: local lab rehearsal script, not an official Azure command. |

| Inspect CI evaluation result | `gh run view <run-id> --log` | CI output stores metric artifacts and pass/fail status. Command type: GitHub CLI workflow verification. |

Implement observability for generative AI applications and agents

Exam Radar

- Official Blueprint Mapping: Examine continuous monitoring in Foundry; Monitor performance metrics, including latency, throughput, and response times; Track and optimize cost metrics, including token consumption and resource usage; Configure detailed logging, tracing, and debugging capabilities for production troubleshooting.
- Domain Weight: Implement generative AI quality assurance and observability represents 10-15% of the official skills measured.
- Core Priority: The exam tests whether the candidate can operate this sub-skill as a concrete Azure workflow inside Implement generative AI quality assurance and observability, not merely identify the service name.
- High Frequency: Expect scenario wording that combines Trace span, Correlation ID, Latency metric, and verification evidence from commands, logs, metrics, or portal state.
- Confusion Alert: A 200 response only proves transport success; quality, cost, trace completeness, and model latency require separate telemetry.
- Scenario Logic: Choose the answer that creates a verifiable dependency chain: configure the object, bind identity or version, run the operation, then prove the resulting state.
- Version Delta: AI-300 combines Azure Machine Learning and Microsoft Foundry under AIOps, so this point must be read as an operational platform task rather than a standalone concept.
- Failure Trigger: Production incidents cannot be reconstructed when retrieval, prompt assembly, model call, tool call, and response serialization lack a shared correlation ID.
- Operational Dependency: The task depends on Trace span, Correlation ID, Latency metric, Token metric, Application log, Alert rule and a validation step that proves the configured state is actually

usable.

- How the Exam Asks It: A GenAI app has intermittent poor answers and high latency, but logs only show HTTP 200 responses.
- How Distractors Are Designed: Distractors monitor only uptime, add more prompt examples, or change the model without trace evidence. These do not reveal retrieval, tool, model, or token-cost bottlenecks.
- Why the Correct Answer Works: The correct answer instruments correlated traces, logs, latency, throughput, token usage, and failure metrics across retrieval, prompt assembly, model calls, and tools.

Atomic Deconstruction - Operational Level

Microscopic technical focus: Monitoring latency, throughput, token consumption, resource usage, logs, traces, and production debugging signals.

Beginner explanation: Observability connects one user request to retrieval, prompt creation, model call, tool calls, token cost, and response. HTTP success is only one small signal.

Operational split for this point: start with Trace span, then verify Correlation ID and Latency metric before trusting any production outcome. The exam is testing whether the candidate can locate the missing dependency, not whether the candidate recognizes every service name in the scenario.

For this knowledge point, the target objects are Trace span, Correlation ID, Latency metric, Token metric, Application log, Alert rule. The exam usually describes one broken link in that chain. The correct answer is the option that restores the missing operational dependency rather than the option that only describes the platform at a high level.

Why-layer: Trace span becomes exam-relevant only when the surrounding dependency chain can run. In this topic, Production incidents cannot be reconstructed when retrieval, prompt assembly, model call, tool call, and response serialization lack a shared correlation ID. The correct configuration matters because it changes the state that controls execution, authorization, resolution, evaluation, or observability; a nearby but unrelated action leaves the same failure mode in place.

Decision tree: if the scenario describes access failure, inspect identity and RBAC before changing compute or code; if it describes unresolved assets, inspect name, version, and scope; if it describes runtime failure, inspect logs, endpoint invocation, metrics, or evaluation output; if it describes quality degradation, inspect data, retrieval, evaluation, and monitoring evidence before changing the model.

Common mistakes: Selecting a familiar Azure service without checking the missing dependency in the scenario. Treating a successful create operation as proof of runtime behavior. Choosing a monitoring action when the scenario asks for configuration or access remediation.

Practice question: A GenAI app has intermittent poor answers and high latency, but logs only show HTTP 200 responses.
A. Instrument correlation IDs and traces across retrieval, prompt assembly, model calls, tool calls, latency,

token usage, and errors.
B. Track only HTTP 200 responses because successful transport proves response quality.
C. Switch model versions before collecting trace evidence.
D. Increase max tokens to troubleshoot latency spikes.

expected answer: A

Explanation: A is correct because GenAI incidents span multiple runtime stages. B ignores quality/cost/tool behavior, C changes the system before diagnosis, and D may increase latency and cost.

The common decision point is: A 200 response only proves transport success; quality, cost, trace completeness, and model latency require separate telemetry. Therefore, read every scenario for the actor, the resource scope, the object version, the network path, the metric threshold, and the expected observable result.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Correlation ID | Trace linkage | Request-level unique ID | Absent unless generated | App instrumentation | Incident cannot be reconstructed |

| Trace span | Operation boundary | Retrieval, prompt, model, tool, response | No span until instrumented | Telemetry SDK or Microsoft Foundry tracing | Latency source is hidden |
| Token metric | Cost signal | Prompt, completion, total tokens | Untracked by default in app logs | Model response telemetry | Cost spike has no owner |

| Alert rule | Operational threshold | Latency, failure, token, safety, quality | No notification | Metric source and action group | Degradation remains dashboard-only |

Step-by-Step Execution Path

1. Execute the operational step.
   curl -H "x-correlation-id: "
   Command type: network/API rehearsal command for the selected app or Azure OpenAI endpoint; confirm URL, header, and API version before use.

Reason: Send a known correlation ID so the full request path can be searched across app logs, traces, and model telemetry.

Checkpoint: The same ID appears in downstream logs or traces.

2. Execute the operational step.
   az monitor log-analytics query --workspace --analytics-query "AppTraces | where CorrelationId == ""

Command type: Azure Monitor Logs verification; confirm workspace schema and query table names in the lab environment.

Reason: Query traces by correlation ID because HTTP status alone cannot show where the request spent time.

Checkpoint: Results show retrieval, prompt, model, tool, and response spans.

3. Execute the operational step.
   az monitor metrics list --resource --metric Latency,TotalTokens
   Command type: Azure Monitor CLI verification; confirm metric names for the selected Azure resource type.

Reason: Inspect latency and token metrics together because slow or expensive responses can come from prompt size, model latency, or tool calls.

Checkpoint: Metrics show time-series latency and token usage.

4. Execute the operational step.
   az monitor metrics alert create --name genai-latency-token-alert --resource-group rg-ai300 --scopes
   Command type: Azure Monitor CLI verification; confirm metric names for the selected Azure resource type.

Reason: Create alerts so degradation is actionable instead of discovered after user complaints.

Checkpoint: Alert rule is enabled and bound to an action group.

Technical Chain

A user, workflow, or deployment command targets Trace span and submits configuration to Azure control plane or a project runtime. Azure validates identity, resource scope, quota, version references, and network reachability because the runtime cannot safely use an object that is not authorized, versioned, reachable, or measurable. The configured object then participates in the runtime path through Correlation ID, Latency metric, Token metric. This sequence works because each object unlocks the next dependency: identity allows access, versioning allows reproducibility, network resolution allows execution, and telemetry allows verification. When the workload executes, telemetry, status output, logs, API response, or evaluation metrics prove whether the chain is complete. If the chain breaks, the failure appears as the operational symptom described in the scenario: Production incidents cannot be reconstructed when retrieval, prompt assembly, model call, tool call, and response serialization lack a shared correlation ID. An incorrect configuration creates the observed failure because it changes a nearby object while leaving the actual missing dependency unresolved.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

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

| Send correlated request | `curl -H "x-correlation-id: <id>" <app-endpoint>` | The same correlation ID appears in downstream traces. Command type: network/API rehearsal command for the selected app or Azure OpenAI endpoint; confirm URL, header, and API version before use. |

| Query traces by correlation ID | `az monitor log-analytics query --workspace <workspace-id> --analytics-query "AppTraces \| where CorrelationId == '<id>'"` |
| Review latency and token metrics | `az monitor metrics list --resource <resource-id> --metric Latency,TotalTokens` | Metrics expose latency and token consumption together. Command type: Azure Monitor CLI verification; confirm metric names for the selected Azure resource type. |

| Verify alert rule | `az monitor metrics alert show --name genai-latency-token-alert --resource-group rg-ai300` | Alert rule exists at the intended scope. Command type: Azure Monitor CLI verification; confirm metric names for the selected Azure resource type. |

Practice Questions

1. A chatbot update improves fluency but starts giving unsupported answers. Which evaluation metric is most directly relevant?
   A. Model registry archive status
   B. Compute cluster idle shutdown time
   C. Storage account access tier
   D. Groundedness or citation faithfulness against the retrieved context

2. A team wants to compare two prompt versions before release using a fixed scenario set. What should they create?
   A. A larger batch scoring output folder
   B. A new public IP address for the workspace
   C. An evaluation dataset with expected criteria and repeatable scoring
   D. A random set of live user messages with no labeling

3. A generative AI application occasionally returns unsafe content. Which control should be reviewed together with evaluation results?
   A. The Azure Machine Learning model registry version order
   B. Content safety filters, harm-category thresholds, and blocked-response telemetry
   C. The number of CPU nodes in a training cluster
   D. The endpoint traffic split for a classical model

4. Users report intermittent slow responses from a RAG application. Which observability data is most useful?
   A. End-to-end traces showing retrieval latency, model latency, token counts, and dependency calls
   B. A list of workspace tags only

C. The previous week's branch names
D. The number of registered data assets

5. A release gate requires evidence that a new agent version handles tool failures correctly. Which test set is most appropriate?
   A. A list of endpoint names
   B. Only happy-path questions with no tool calls
   C. A storage firewall configuration export
   D. Scenario evaluations that inject tool errors, timeout cases, and expected fallback behavior

6. A monitoring dashboard shows higher token cost but stable user volume. What should the engineer inspect first?
   A. Compute cluster min node count
   B. Key Vault purge protection only
   C. Prompt length, retrieved context size, conversation history, and model selection changes
   D. The number of model registry aliases

7. A support team needs to investigate one harmful answer reported by a user. What telemetry is most important?
   A. The Azure subscription display name
   B. Correlated request trace with prompt, retrieved context reference, model response, safety result, and user session metadata allowed by policy
   C. The number of Bicep modules in the repository
   D. The training compute idle timeout value

8. A new retrieval strategy improves answer relevance in offline tests but worsens production abandonment rate. What should the team compare?
   A. Offline evaluation scores with production telemetry such as latency, no-answer rate, feedback, and task completion
   B. Only the number of files in the repository
   C. Only the Azure region name
   D. The model registry creation date

9. An evaluator reports that answers are correct but too verbose for customer support workflows. Which evaluation dimension should be added?
   A. Response conciseness or instruction adherence criteria
   B. Storage replication lag
   C. GPU driver version
   D. Endpoint DNS zone count

10. A team wants audit-ready evidence for why a prompt was released. Which artifacts should be retained?
    A. A screenshot of the browser home page
    B. Only the final answer from one manual test

C. Prompt version, evaluation dataset, metric results, approval record, and deployment trace
D. A renamed local folder

Optimize generative AI systems and model performance

Optimize retrieval-augmented generation performance and accuracy

Exam Radar

- Official Blueprint Mapping: Optimize retrieval performance by tuning similarity thresholds, chunk sizes, and retrieval strategies; Select and fine-tune embedding models for domain-specific use cases and accuracy improvements; Implement and optimize hybrid search approaches combining semantic and keyword-based retrieval; Evaluate and improve RAG system performance by using relevance metrics and A/B testing frameworks.
- Domain Weight: Optimize generative AI systems and model performance represents 10-15% of the official skills measured.
- Core Priority: The exam tests whether the candidate can operate this sub-skill as a concrete Azure workflow inside Optimize generative AI systems and model performance, not merely identify the service name.
- High Frequency: Expect scenario wording that combines Chunking policy, Vector index, Embedding model, and verification evidence from commands, logs, metrics, or portal state.
- Confusion Alert: Fine-tuning is the wrong first fix when the root cause is missing retrieval coverage or poorly ranked source chunks.
- Scenario Logic: Choose the answer that creates a verifiable dependency chain: configure the object, bind identity or version, run the operation, then prove the resulting state.
- Version Delta: AI-300 combines Azure Machine Learning and Microsoft Foundry under AIOps, so this point must be read as an operational platform task rather than a standalone concept.
- Failure Trigger: Answers become ungrounded when thresholds exclude relevant chunks, chunking breaks context, or embedding vectors are generated with a different model than the index.
- Operational Dependency: The task depends on Chunking policy, Vector index, Embedding model, Similarity threshold, Hybrid search profile, A/B test variant and a validation step that proves the configured state is actually usable.
- How the Exam Asks It: A RAG app gives ungrounded answers or misses exact product and policy terms, and the team must decide whether to tune retrieval or fine-tune the model.
- How Distractors Are Designed: Distractors fine-tune immediately, increase max tokens, or switch to a larger model. These do not fix missing context, chunking errors, or poor ranking.
- Why the Correct Answer Works: The correct answer evaluates retrieval quality, tunes chunking, thresholds, embedding model, hybrid search, and reranking before considering fine-tuning.

Atomic Deconstruction - Operational Level

Microscopic technical focus: Tuning chunk sizes, similarity thresholds, retrieval strategies, embedding models, hybrid search, relevance metrics, and A/B tests.

Beginner explanation: RAG quality usually fails before the model answers: the wrong chunks are retrieved, ranked, filtered, or packed into the prompt.

Operational split for this point: start with Chunking policy, then verify Vector index and Embedding model before trusting any production outcome. The exam is testing whether the candidate can locate the missing dependency, not whether the candidate recognizes every service name in the scenario.

For this knowledge point, the target objects are Chunking policy, Vector index, Embedding model, Similarity threshold, Hybrid search profile, A/B test variant. The exam usually describes one broken link in that chain. The correct answer is the option that restores the missing operational dependency rather than the option that only describes the platform at a high level.

Why-layer: Chunking policy becomes exam-relevant only when the surrounding dependency chain can run. In this topic, Answers become ungrounded when thresholds exclude relevant chunks, chunking breaks context, or embedding vectors are generated with a different model than the index. The correct configuration matters because it changes the state that controls execution, authorization, resolution, evaluation, or observability; a nearby but unrelated action leaves the same failure mode in place.

Decision tree: if the scenario describes access failure, inspect identity and RBAC before changing compute or code; if it describes unresolved assets, inspect name, version, and scope; if it describes runtime failure, inspect logs, endpoint invocation, metrics, or evaluation output; if it describes quality degradation, inspect data, retrieval, evaluation, and monitoring evidence before changing the model.

Common mistakes: Choosing fine-tuning when retrieved context is missing or poorly ranked. Changing the embedding model without rebuilding the vector index. Raising top-k without measuring latency and context noise.

Practice question: A RAG app gives ungrounded answers or misses exact product and policy terms, and the team must decide whether to tune retrieval or fine-tune the model.
A. Evaluate retrieval quality, tune chunking/thresholds/embedding or hybrid search, and compare variants before considering fine-tuning.
B. Fine-tune immediately because all ungrounded answers indicate model knowledge gaps.
C. Increase top-k without measuring context noise, latency, or groundedness.
D. Change the embedding model while keeping the old vector index.

expected answer: A

Explanation: A is correct because RAG failures often occur before generation. B solves the wrong layer, C may add irrelevant context, and D creates incompatible vector comparisons.

The common decision point is: Fine-tuning is the wrong first fix when the root cause is missing retrieval coverage or poorly ranked source chunks. Therefore, read every scenario for the actor, the resource scope,

the object version, the network path, the metric threshold, and the expected observable result.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Chunking policy | Window and overlap | Token/character size plus overlap | Default splitter | Document structure and tokenizer | Relevant evidence is split or diluted |

| Embedding model | Vector space | Model-specific dimension | Existing index vector space | Re-embedding pipeline | Query vectors cannot compare correctly |
| Similarity threshold | Candidate filter | Engine-specific score range | Untuned default | Evaluation dataset | Relevant chunks excluded or noisy chunks included |

| Hybrid search | Ranking blend | Vector, keyword, semantic reranker | Single retrieval mode | Search index fields | Exact identifiers or paraphrases are missed |

Step-by-Step Execution Path

1. Execute the operational step.
   python rag_eval.py --dataset evaluations/rag-gold.jsonl --index policies-v1 --metrics recall_at_k mrr groundedness
   Command type: local lab rehearsal script, not an official Azure command.

Reason: Establish a baseline so changes can be judged by retrieval and answer quality instead of anecdotal samples.

Checkpoint: Report contains recall@k, MRR, and groundedness for the current index.

2. Execute the operational step.
   python build_index.py --source data/policies --chunk-size 800 --chunk-overlap 120 --embedding-model text-embedding-3-large --index policies-v2
   Command type: local lab rehearsal script, not an official Azure command.

Reason: Rebuild the index after chunk or embedding changes because stored vectors and chunk boundaries define retrieval behavior.

Checkpoint: Index metadata shows expected document count and vector dimension.

3. Execute the operational step.
   python rag_eval.py --dataset evaluations/rag-gold.jsonl --index policies-v2 --retrieval hybrid --semantic-rerank true
   Command type: local lab rehearsal script, not an official Azure command.

Reason: Test hybrid retrieval when exact IDs and semantic paraphrases both matter.

Checkpoint: Metrics improve without unacceptable latency growth.

4. Execute the operational step.
   python route_variant.py --variant-a policies-v1 --variant-b policies-v2 --split 50 --metric groundedness
   Command type: local lab rehearsal script, not an official Azure command.

Reason: Run a controlled variant test so production decision uses measured quality, latency, and token impact.

Checkpoint: Telemetry is segmented by variant ID and shows winning configuration.

Technical Chain

A user, workflow, or deployment command targets Chunking policy and submits configuration to Azure control plane or a project runtime. Azure validates identity, resource scope, quota, version references, and network reachability because the runtime cannot safely use an object that is not authorized, versioned, reachable, or measurable. The configured object then participates in the runtime path through Vector index, Embedding model, Similarity threshold. This sequence works because each object unlocks the next dependency: identity allows access, versioning allows reproducibility, network resolution allows execution, and telemetry allows verification. When the workload executes, telemetry, status output, logs, API response, or evaluation metrics prove whether the chain is complete. If the chain breaks, the failure appears as the operational symptom described in the scenario: Answers become ungrounded when thresholds exclude relevant chunks, chunking breaks context, or embedding vectors are generated with a different model than the index. An incorrect configuration creates the observed failure because it changes a nearby object while leaving the actual missing dependency unresolved.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Measure retrieval baseline | `python rag_eval.py --dataset evaluations/rag-gold.jsonl --index policies-v1 --metrics recall_at_k mrr groundedness` | Baseline report contains retrieval and groundedness metrics. Command type: local lab rehearsal script, not an official Azure command. |

| Inspect rebuilt index metadata | `python inspect_index.py --index policies-v2 --show-metadata` | Chunk count, embedding model, and vector dimension match the intended configuration. Command type: local lab rehearsal script, not an official Azure command. |
| Compare hybrid retrieval | `python rag_eval.py --dataset evaluations/rag-gold.jsonl --index policies-v2 --retrieval hybrid --semantic-rerank true` | Metrics improve without unacceptable latency growth. Command type: local lab rehearsal script, not an official Azure command. |

| Check variant telemetry | `python compare_variants.py --metric groundedness --variants policies-v1 policies-v2` | Telemetry is segmented by variant and identifies the better configuration. Command type: local lab rehearsal script, not an official Azure command. |

Implement advanced fine-tuning and model customization

Exam Radar

- Official Blueprint Mapping: Design and implement advanced fine-tuning methods; Create and manage synthetic data for fine-tuning; Monitor and optimize fine-tuned model performance; Manage a fine-tuned model from development through production deployment.
- Domain Weight: Optimize generative AI systems and model performance represents 10-15% of the official skills measured.
- Core Priority: The exam tests whether the candidate can operate this sub-skill as a concrete Azure workflow inside Optimize generative AI systems and model performance, not merely identify the service name.
- High Frequency: Expect scenario wording that combines Fine-tuning dataset, Synthetic data, Validation split, and verification evidence from commands, logs, metrics, or portal state.
- Confusion Alert: Fine-tuning should target behavior, structure, or domain patterns after retrieval and prompting limits are proven with evaluation evidence.
- Scenario Logic: Choose the answer that creates a verifiable dependency chain: configure the object, bind identity or version, run the operation, then prove the resulting state.
- Version Delta: AI-300 combines Azure Machine Learning and Microsoft Foundry under AIOps, so this point must be read as an operational platform task rather than a standalone concept.
- Failure Trigger: A customized model regresses in production when synthetic examples contain label noise, duplicate patterns, unsafe outputs, or no validation baseline.
- Operational Dependency: The task depends on Fine-tuning dataset, Synthetic data, Validation split, Fine-tuned model, Deployment version, Performance monitor and a validation step that proves the configured state is actually usable.
- How the Exam Asks It: Prompting and retrieval are stable, but the model still fails a repeated format, domain language, or task behavior requirement.
- How Distractors Are Designed: Distractors fine-tune with unreviewed synthetic data, skip validation, or use fine-tuning to add missing facts. These introduce regression or solve the wrong problem.
- Why the Correct Answer Works: The correct answer validates curated training and validation data, uses synthetic data only after quality checks, evaluates the customized model, and promotes it with monitoring.

Atomic Deconstruction - Operational Level

Microscopic technical focus: Preparing supervised data, synthetic data, validation sets, fine-tuned model versions, monitoring, and production promotion.

Beginner explanation: Fine-tuning changes model behavior. It is appropriate after prompting and retrieval limits are proven, not as a shortcut for missing knowledge.

Operational split for this point: start with Fine-tuning dataset, then verify Synthetic data and Validation split before trusting any production outcome. The exam is testing whether the candidate can locate the missing dependency, not whether the candidate recognizes every service name in the scenario.

For this knowledge point, the target objects are Fine-tuning dataset, Synthetic data, Validation split, Fine-tuned model, Deployment version, Performance monitor. The exam usually describes one broken link in that chain. The correct answer is the option that restores the missing operational dependency rather than the option that only describes the platform at a high level.

Why-layer: Fine-tuning dataset becomes exam-relevant only when the surrounding dependency chain can run. In this topic, A customized model regresses in production when synthetic examples contain label noise, duplicate patterns, unsafe outputs, or no validation baseline. The correct configuration matters because it changes the state that controls execution, authorization, resolution, evaluation, or observability; a nearby but unrelated action leaves the same failure mode in place.

Decision tree: if the scenario describes access failure, inspect identity and RBAC before changing compute or code; if it describes unresolved assets, inspect name, version, and scope; if it describes runtime failure, inspect logs, endpoint invocation, metrics, or evaluation output; if it describes quality degradation, inspect data, retrieval, evaluation, and monitoring evidence before changing the model.

Common mistakes: Selecting a familiar Azure service without checking the missing dependency in the scenario. Treating a successful create operation as proof of runtime behavior. Choosing a monitoring action when the scenario asks for configuration or access remediation.

Practice question: Prompting and retrieval are stable, but the model still fails a repeated format, domain language, or task behavior requirement.
A. Validate curated training and holdout data, inspect synthetic examples, evaluate the customized model, and monitor the promoted deployment.
B. Generate a large synthetic dataset and fine-tune without manual quality review.
C. Use fine-tuning to add missing facts that should have been retrieved from source documents.
D. Skip validation because training loss is enough to prove production readiness.

expected answer: A

Explanation: A is correct because customization requires clean examples, holdout evaluation, and production monitoring. B risks noisy behavior, C misuses fine-tuning for knowledge injection, and D ignores overfitting and safety regression.

The common decision point is: Fine-tuning should target behavior, structure, or domain patterns after retrieval and prompting limits are proven with evaluation evidence. Therefore, read every scenario for the actor, the resource scope, the object version, the network path, the metric threshold, and the expected observable result.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Fine-tuning dataset | Example quality | Prompt/completion or chat format | Unusable until schema-valid | Model fine-tuning format | Training fails or learns noisy behavior |

| Synthetic data | Augmentation source | Generated examples with review status | Untrusted until inspected | Quality and safety review | Model learns unrealistic patterns |
| Validation split | Holdout evidence | Representative labeled examples | Absent until separated | Evaluation workflow | Overfit model appears successful |

| Fine-tuned deployment | Promotion state | Candidate, staging, production | Not serving until deployed | Endpoint and monitor | Regression reaches users without rollback path |

Step-by-Step Execution Path

1. Execute the operational step.
   python validate_finetune_data.py --train data/train.jsonl --validation data/valid.jsonl
   Command type: local lab rehearsal script, not an official Azure command.

Reason: Validate schema and split before training because malformed or duplicated examples create unreliable customization.

Checkpoint: Report shows schema pass, duplicate rate, and label distribution.

2. Execute the operational step.
   python inspect_synthetic_data.py --input data/synthetic.jsonl --report reports/synthetic-quality.json
   Command type: local lab rehearsal script, not an official Azure command.

Reason: Inspect synthetic data because generated examples can amplify wrong labels or unsafe patterns.

Checkpoint: Quality report flags duplicates, leakage, and safety concerns.

3. Execute the operational step.
   python evaluate.py --dataset evaluations/finetune-valid.jsonl --model
   Command type: local lab rehearsal script, not an official Azure command.

Reason: Evaluate the customized model on holdout cases so improvement is proven beyond the training examples.

Checkpoint: Evaluation report shows target behavior improves without safety regression.

4. Execute the operational step.
   az monitor metrics list --resource --metric Latency,TotalCalls,Failures
   Command type: Azure Monitor CLI verification; confirm metric names for the selected Azure resource type.

Reason: Monitor the fine-tuned deployment because customization can change latency, failure rate, and output shape.

Checkpoint: Metrics show stable production behavior after promotion.

Technical Chain

A user, workflow, or deployment command targets Fine-tuning dataset and submits configuration to Azure control plane or a project runtime. Azure validates identity, resource scope, quota, version references, and network reachability because the runtime cannot safely use an object that is not authorized, versioned, reachable, or measurable. The configured object then participates in the runtime path through Synthetic data, Validation split, Fine-tuned model. This sequence works because each object unlocks the next dependency: identity allows access, versioning allows reproducibility, network resolution allows execution, and telemetry allows verification. When the workload executes, telemetry, status output, logs, API response, or evaluation metrics prove whether the chain is complete. If the chain breaks, the failure appears as the operational symptom described in the scenario: A customized model regresses in production when synthetic examples contain label noise, duplicate patterns, unsafe outputs, or no validation baseline. An incorrect configuration creates the observed failure because it changes a nearby object while leaving the actual missing dependency unresolved.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

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

| Validate fine-tuning data | `python validate_finetune_data.py --train data/train.jsonl --validation data/valid.jsonl` | Report shows schema pass, duplicate rate, and label distribution. Command type: local lab rehearsal script, not an official Azure command. |

| Inspect synthetic examples | `python inspect_synthetic_data.py --input data/synthetic.jsonl --report reports/synthetic-quality.json` | Quality report flags duplicates, leakage, and safety concerns. Command type: local lab rehearsal script, not an official Azure command. |

| Evaluate customized model | `python evaluate.py --dataset evaluations/finetune-valid.jsonl --model <fine-tuned-model>` | Holdout results prove improvement without safety regression. Command type: local lab rehearsal script, not an official Azure command. |

| Review deployment metrics | `az monitor metrics list --resource <deployment-resource-id> --metric Latency,TotalCalls,Failures` | Production telemetry remains stable after promotion. Command type: Azure Monitor CLI verification; confirm metric names for the selected Azure resource type. |

Practice Questions

1. A RAG application returns answers that omit relevant policy details even though the documents exist in the index. What should be inspected first?
   A. Chunking strategy, embedding coverage, retrieval top-k, filters, and reranking behavior
   B. Azure Machine Learning compute idle timeout only
   C. The name of the GitHub repository
   D. Key Vault soft-delete status

2. Search results are semantically close but frequently miss exact product codes. Which retrieval improvement is most appropriate?
   A. Lowering all safety thresholds
   B. Hybrid search that combines vector retrieval with keyword or lexical matching
   C. Deleting the index and relying only on the model's memory
   D. Increasing endpoint traffic to the current deployment

3. A team changes the embedding model for an existing vector index and retrieval starts failing. What must be validated?
   A. Vector field dimensions and re-embedding of indexed content with the selected embedding model
   B. The number of pull request reviewers
   C. The display name of the Foundry project
   D. The batch endpoint schedule

4. A prompt includes ten retrieved chunks, but answers are slower and less focused. Which tuning action is most appropriate?
   A. Rotate storage account keys
   B. Evaluate smaller top-k values, chunk sizes, and reranking thresholds against relevance and latency
   C. Increase all generated response lengths without testing
   D. Disable all observability

5. A base foundation model performs poorly on a specialized classification style even with strong prompts and examples. What should be considered next?
   A. Fine-tuning or model customization using representative, governed training examples
   B. Removing the evaluation dataset
   C. Moving all prompts out of source control
   D. Disabling content filters

6. A fine-tuned model appears better on training examples but worse on new customer cases. What is the likely issue?
   A. Too many private endpoints in the workspace
   B. Overfitting or unrepresentative fine-tuning data
   C. A missing model registry description
   D. A storage account with geo-redundant replication

7. A team wants to reduce hallucination in a support assistant before trying fine-tuning. Which action should usually come first?
   A. Improve retrieval grounding, source selection, prompt constraints, and groundedness evaluation
   B. Delete all citations from responses
   C. Increase response creativity settings for every query
   D. Remove the retrieval index from the application

8. A/B testing shows model B has slightly higher quality but doubles latency and token cost. What should the release decision consider?
   A. Only the storage account region
   B. Quality, latency, cost, user impact, and service-level objectives together
   C. Only the number of prompt files changed
   D. Only the model display name

9. A retrieval index contains stale documents that conflict with newer policies. What optimization should be prioritized?
   A. Creating a new workspace tag
   B. Increasing model temperature
   C. Adding more endpoint replicas without changing data
   D. Index freshness controls, source ingestion validation, and metadata-based filtering

10. A team plans to fine-tune a model using customer conversation logs. What must be handled before training?
    A. Deleting the prompt repository
    B. Disabling all model monitoring
    C. Data consent, privacy review, sensitive-data filtering, representative labeling, and evaluation split design
    D. Publishing the raw logs as a public data asset

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 Design and implement an MLOps infrastructure, Implement machine learning model lifecycle and operations, Design and implement a GenAIOps infrastructure.

- 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 Machine Learning Operations MLOps Engineer Associate exam who want a structured, exam-aligned review resource. It is especially useful for professionals who need to connect the exam's knowledge points with practical responsibilities, business context, and operational decision-making.

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

Design and implement an MLOps infrastructure

Q1. Correct answer: B
Explanation: Private endpoints and private networking address the access-path requirement for the workspace and dependent services. Tags do not restrict ingress, larger compute does not solve network exposure, and local storage breaks governed MLOps data flow.

Q2. Correct answer: A
Explanation: A datastore read failure is usually caused by identity, credential, or storage authorization issues. Model versions, prompt temperature, and endpoint autoscale are unrelated to workspace datastore access.

Q3. Correct answer: D
Explanation: Infrastructure as code makes workspace configuration repeatable, reviewable, and parameterized per environment. Portal-only setup is difficult to audit, personal workspaces fragment governance, and MLflow metrics do not provision infrastructure.

Q4. Correct answer: C
Explanation: Azure Machine Learning assets provide reusable, versioned objects for data, environments, components, and models. Availability tests, policy exemptions, and prompt variants do not provide the core workspace artifact-sharing mechanism.

Q5. Correct answer: C
Explanation: Versioning the environment definition preserves the runtime image and dependencies that affect training behavior. Display names, minimum node count, and telemetry sampling do not define the Python or container runtime used by the job.

Q6. Correct answer: A
Explanation: Azure Machine Learning compute clusters are designed for job execution with autoscale and idle scale-down behavior. Online endpoints serve inference, Key Vault governs secrets, and Azure AI Search configurations support retrieval.

Q7. Correct answer: D
Explanation: Training jobs that cannot pull images point to Azure Container Registry authorization or network path problems. Semantic ranking, prompt tags, and evaluation thresholds do not control container image retrieval.

Q8. Correct answer: B
Explanation: RBAC and policy govern who can create compute and which SKUs are allowed. Endpoint timeouts, model temperature, and labels do not prevent unauthorized infrastructure creation.

Q9. Correct answer: C
Explanation: Secret retrieval failures after private networking changes usually involve Key Vault network access or identity permissions. Prompt examples, endpoint traffic, and vector dimensions are separate concerns.

Q10. Correct answer: D
Explanation: Pull requests and deployment logs show review, approval, and applied infrastructure changes. Screenshots are weak point-in-time evidence, while prediction responses and prompt names do not prove infrastructure governance.

Implement machine learning model lifecycle and operations

Q1. Correct answer: B
Explanation: Pipeline jobs with reusable components formalize repeatable ML workflows. Manual notebooks and unmanaged VMs reduce reproducibility, while anonymous storage weakens data governance.

Q2. Correct answer: D
Explanation: Registration with metrics enables controlled promotion and traceability. Deleting versions harms rollback, compute scaling does not validate quality, and disabling telemetry weakens production operations.

Q3. Correct answer: C
Explanation: Managed online endpoints can split traffic across deployments for staged rollout. Batch inference is offline scoring, data asset versioning governs data, and private endpoints govern networking.

Q4. Correct answer: A
Explanation: Batch endpoints or scheduled pipelines are designed for offline large-scale scoring. Real-time endpoints are for low-latency requests, prompt A/B testing targets GenAI behavior, and laptop scoring is not operationally reliable.

Q5. Correct answer: D
Explanation: Latency investigations begin with endpoint metrics and logs that reveal request behavior and resource pressure. Display names, repository file counts, and Key Vault retention do not explain inference latency directly.

Q6. Correct answer: B
Explanation: Versioned models and separate deployments make rollback traceable and executable. Overwriting the latest file, deleting tracking data, and anonymous storage undermine controlled operations.

Q7. Correct answer: C
Explanation: MLflow tracking captures run evidence needed to compare and reproduce experiments. It does not replace RBAC, create private endpoints, or tune prompts.

Q8. Correct answer: A
Explanation: Drift and model monitoring compare production signals against baselines. Workspace names, Key Vault SKU changes, and branch names do not detect statistical changes in model input behavior.

Q9. Correct answer: D
Explanation: Failed health probes usually indicate scoring script, dependency, container startup, or deployment state problems. Reader assignments, Foundry safety policy, and vector index settings do not explain a classic model container startup failure.

Q10. Correct answer: B
Explanation: A quality gate belongs after evaluation evidence is produced and before traffic moves to the candidate. Deleting the endpoint, waiting for user complaints, or documenting firewall rules does not prevent bad releases.

Design and implement a GenAIOps infrastructure

Q1. Correct answer: C
Explanation: Foundry hubs/projects with RBAC and managed identities support governed GenAIOps separation. Shared accounts, local-only prompts, and disabled controls weaken auditability and access boundaries.

Q2. Correct answer: D
Explanation: A 403 response indicates authorization failure, so role assignment and scope are the first checks. Retrieval chunk count, prompt formatting, and batch output folders target different problem areas.

Q3. Correct answer: A
Explanation: Separate deployment and controlled release allow validation before production cutover. Direct replacement, deleting telemetry, and removing filters all increase operational risk.

Q4. Correct answer: B
Explanation: Prompt versioning plus evaluation evidence lets teams compare behavior and roll back. Portal-only edits and anonymous files reduce traceability, while GPU compute is unrelated to prompt lifecycle control.

Q5. Correct answer: C
Explanation: Throttling is tied to throughput, quota, token consumption, and request rate. Storage retention, MLflow tags, and PR counts do not reveal model-serving capacity pressure.

Q6. Correct answer: D
Explanation: Managed identity and Key Vault avoid hard-coded secrets and support auditable access. Hard-coded keys, committed secrets, and prompt-embedded secrets are insecure and hard to rotate.

Q7. Correct answer: A
Explanation: Connectivity after private networking depends on DNS, private endpoint, firewall, and routing behavior. Model cards, dataset class balance, and prompt example count do not explain subnet-specific access failures.

Q8. Correct answer: B
Explanation: Pull request review and protected environments create a controlled approval path. Direct pushes, manual copy-and-paste, and deleting history remove review evidence.

Q9. Correct answer: C
Explanation: Parameterized configuration preserves consistency where needed and lets capacity vary by environment. Hard-coding, random names, and chat-only configuration are difficult to govern.

Q10. Correct answer: D
Explanation: The active prompt depends on what was built, deployed, and selected at runtime. Compute cluster size, storage replication, and model registry descriptions do not prove which prompt version the app executed.

Implement generative AI quality assurance and observability

Q1. Correct answer: D
Explanation: Unsupported answers are a groundedness or faithfulness issue. Compute shutdown, storage access tier, and registry archive state do not measure whether responses align with source context.

Q2. Correct answer: C
Explanation: A repeatable evaluation dataset allows comparable scoring across prompt versions. Public IPs and output folders are unrelated, while unlabeled random messages cannot support controlled release decisions.

Q3. Correct answer: B
Explanation: Unsafe content requires reviewing safety filters, thresholds, and telemetry showing blocked or allowed responses. ML model registry order, CPU nodes, and classical endpoint traffic do not directly govern GenAI safety output.

Q4. Correct answer: A
Explanation: RAG latency comes from multiple stages, so traces and dependency metrics reveal where time is spent. Tags, branch names, and data asset counts do not isolate runtime bottlenecks.

Q5. Correct answer: D
Explanation: Tool-failure behavior must be tested with scenarios that exercise those failures and verify fallback logic. Happy-path-only tests and configuration lists do not prove resilient agent behavior.

Q6. Correct answer: C
Explanation: Token cost is driven by prompt, context, history, and model behavior. Key Vault settings, compute min nodes, and registry aliases do not explain token growth.

Q7. Correct answer: B
Explanation: A correlated trace provides the evidence needed to reconstruct the response path within privacy and policy limits. Subscription names, module counts, and idle timeouts do not explain the reported answer.

Q8. Correct answer: A
Explanation: Release quality must combine offline scores with production behavior. Repository file counts, region names, and registry dates do not show whether users complete tasks successfully.

Q9. Correct answer: A
Explanation: Verbosity is an instruction-adherence or conciseness issue and should be measured explicitly. Storage replication, GPU drivers, and DNS zone counts do not evaluate response style.

Q10. Correct answer: C
Explanation: Audit-ready release evidence links the prompt version to evaluation results, approval, and deployment. A single manual answer, screenshot, or folder rename cannot support a release decision.

Optimize generative AI systems and model performance

Q1. Correct answer: A
Explanation: Missing relevant details usually points to retrieval design and ranking behavior. Compute timeout, repository name, and Key Vault soft delete do not determine which passages are retrieved.

Q2. Correct answer: B
Explanation: Hybrid search helps when exact terms such as product codes matter alongside semantic similarity. Lowering safety thresholds, deleting the index, and shifting traffic do not fix exact-match retrieval gaps.

Q3. Correct answer: A
Explanation: Embedding model changes can alter vector dimensions and require re-indexing. Reviewer counts, project display names, and batch schedules do not fix vector incompatibility.

Q4. Correct answer: B
Explanation: Retrieval context size affects relevance, latency, and token cost, so tuning top-k, chunking, and reranking with evaluation is appropriate. Disabling observability, blindly lengthening answers, or rotating keys does not optimize retrieval quality.

Q5. Correct answer: A
Explanation: When prompting is insufficient for a specialized pattern, fine-tuning with representative examples can be appropriate. Removing evaluation, abandoning source control, or disabling filters increases risk without improving the learned behavior.

Q6. Correct answer: B
Explanation: Better training performance but worse generalization points to overfitting or poor data representativeness. Networking, descriptions, and replication do not explain this evaluation pattern.

Q7. Correct answer: A
Explanation: Hallucination in support scenarios is often best addressed first through grounding and evaluation. Deleting citations, increasing creativity, or removing retrieval usually makes unsupported answers harder to control.

Q8. Correct answer: B
Explanation: Production optimization is a tradeoff across quality, latency, cost, and SLOs. Display names, file counts, and storage region alone do not determine release suitability.

Q9. Correct answer: D
Explanation: Stale or conflicting documents require ingestion freshness and filtering controls. Temperature, replicas, and tags do not remove obsolete context from retrieval results.

Q10. Correct answer: C
Explanation: Fine-tuning with customer logs requires governance, privacy, cleaning, labeling, and evaluation

design before training. Deleting prompts, disabling monitoring, or publishing raw logs creates major operational and compliance risk.