Exam Code: DP-700
Exam Name: DP-700: Microsoft Certified: Fabric Data Engineer Associate Training Course
Certification: Microsoft Certified: Fabric Data Engineer Associate
Vendor: Microsoft

DP-700 Training Course

DP-700: Microsoft Certified: Fabric Data Engineer Associate Training Course

Structured Learning & Certification Preparation

Table of Contents

Introduction

This study pack is designed to support preparation for the Microsoft Certified: Fabric Data Engineer Associate exam through a clear, knowledge-point-driven structure. It brings the exam scope into one place so you can review Implement and manage an analytics solution, Ingest and transform data, Monitor and optimize an analytics solution in the same order you are expected to master them.

The material is organized around 3 official blueprint domains, with each section keeping the detailed explanation content intact and pairing it with mapped practice questions. A practical way to use this pack is to move in a repeatable study, practice, and review cycle: study the explanation first, answer the related questions, then check the answer attachment to confirm where your understanding is already strong and where it still needs reinforcement.

About This Training / Certification

Microsoft Certified: Fabric Data 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

- Implement and manage an analytics solution
    - Configure Microsoft Fabric workspace settings for Spark, OneLake, Dataflows Gen2, and domains
    - Implement lifecycle management with Git integration, database projects, and deployment pipelines
    - Configure security and governance across workspace, item, row, column, object, file, labels, endorsement, and audit logs
    - Orchestrate processes with Dataflows Gen2, pipelines, notebooks, schedules, triggers, parameters, and dynamic expressions
- Ingest and transform data
    - Design and implement full, incremental, dimensional, and streaming loading patterns
    - Ingest and transform batch data with Lakehouse, Warehouse, Dataflows Gen2, notebooks, KQL, T-SQL, shortcuts, mirroring, and pipelines
    - Transform data with SQL, PySpark, and KQL while preserving schema, quality, and analytical intent
- Monitor and optimize an analytics solution

- Identify and resolve ingestion, transformation, shortcut, permission, refresh, and orchestration errors
- Optimize Lakehouse tables, pipelines, warehouses, Eventstreams, Eventhouses, Spark jobs, and queries
- Monitor analytics solutions with Fabric run history, metrics, audit evidence, and operational readiness criteria

---

Detailed Knowledge Explanation

Implement and manage an analytics solution

Configure Microsoft Fabric workspace settings for Spark, OneLake, Dataflows Gen2, and domains

Exam Radar

- Core Priority: Workspace settings define the default execution and governance envelope before any lakehouse, warehouse, pipeline, notebook, or dataflow runs. DP-700 scenarios often ask which workspace-level control should be changed when every item in the workspace inherits the same Spark runtime, OneLake behavior, domain association, or Dataflows Gen2 behavior.
- High Frequency: Expect scenario questions where a team has a correct notebook or pipeline but receives inconsistent execution, storage, or governance behavior because the workspace defaults were not aligned first.
- Confusion Alert: A common distractor is changing item-level settings when the symptom is caused by a workspace-level default. Another distractor is changing tenant settings when the question limits the scope to one workspace.
- Scenario Logic: If multiple engineering artifacts in the same workspace need the same Spark runtime or storage policy, inspect the workspace setting before editing each artifact. If only one notebook or item is affected, item-level configuration becomes more plausible.
- Version Delta: The current Microsoft study guide lists Spark workspace settings, domain workspace settings, OneLake workspace settings, and Dataflows Gen2 workspace settings under this domain as of April 20, 2026.
- Failure Trigger: Misaligned settings produce repeated Spark session configuration drift, wrong domain classification, unsupported shortcut behavior, or Dataflow Gen2 staging and refresh inconsistency.
- Operational Dependency: Workspace admins or users with sufficient workspace permissions must be able to access workspace settings; tenant-level Fabric settings may also constrain what a

workspace can enable.

- How the Exam Asks It: The stem usually gives a repeated behavior across several Fabric items and asks for the lowest-scope configuration change.
- How Distractors Are Designed: Wrong options often use a lakehouse table property, notebook code change, or pipeline activity edit even though the issue begins before item execution.
- Why the Correct Answer Works: The correct answer changes the shared control object that owns the repeated behavior.

Atomic Deconstruction - Operational Level

Fabric workspace settings act as the control-plane starting point for data engineering work. Spark settings determine runtime defaults that affect notebook and Spark job execution. OneLake settings affect how storage and shortcuts behave inside the workspace boundary. Domain settings connect workspace content to enterprise data-domain organization. Dataflows Gen2 settings affect data preparation behavior and refresh handling.

The operational rule is to identify ownership of the behavior. A Spark runtime mismatch is not fixed by changing a SQL warehouse. A domain governance issue is not fixed by optimizing a Delta table. A OneLake shortcut problem is not solved by changing a pipeline retry count. Each setting satisfies a different dependency: execution runtime, storage namespace, governance classification, or data-preparation behavior.

Skipping workspace configuration creates downstream noise. Notebooks may run with unexpected defaults, artifacts may be organized outside the intended domain, and refresh settings may vary by author. The exam rewards the candidate who checks the controlling workspace object before tuning a downstream artifact.

Component Specifications

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

| ------------------------------ | -------------------------- | -- | -- | -- |
| Fabric workspace | Spark settings | Runtime, environment, capacity behavior supported by Fabric | Workspace default | Workspace admin rights and Fabric tenant enablement | Notebooks run with inconsistent runtime behavior |

| Fabric workspace | Domain assignment | Available Fabric domains | Unassigned or tenant default | Domain governance configuration | Items are not organized under the expected data domain |
| OneLake workspace configuration | Storage and shortcut behavior | Supported OneLake options | Workspace inherited behavior | OneLake availability and workspace permissions | Shortcut resolution or storage access errors persist |

| Dataflows Gen2 workspace setting | Refresh and staging behavior | Supported Dataflows Gen2 options | Service defaults | Data Factory workload availability | Dataflow refresh behavior does not match workspace standards |

Step-by-Step Execution Path

1. Open the Fabric portal and navigate to Workspaces > target workspace > Workspace settings. This is first because the scenario symptom applies to the whole workspace.
2. Inspect Spark settings, OneLake settings, domain settings, and Dataflows Gen2 settings. This separates execution, storage, governance, and preparation dependencies before changing individual items.
3. Confirm the user has workspace admin or equivalent permission. A missing permission can make the correct setting invisible and mislead the operator into editing downstream artifacts.
4. Apply the setting that owns the symptom. Use Spark settings for runtime defaults, domain settings for domain association, OneLake settings for storage behavior, and Dataflows Gen2 settings for dataflow behavior.
5. Re-run or refresh one representative item and compare the observed behavior with the expected setting. This validates inheritance without creating broad production churn.

Verification evidence can include official Fabric portal evidence, workspace setting screenshots, item execution status, or supported management API evidence when available for the relevant Fabric version.

Technical Chain

The workspace setting is stored as control-plane metadata. When a notebook, dataflow, pipeline, or lakehouse operation starts, Fabric resolves workspace context before the item operation executes. If the workspace default supplies the runtime or governance property, the item receives that behavior without every artifact restating it. If the default is wrong, each downstream artifact can appear broken even though the item definition is valid. Correcting the workspace metadata changes the dependency that the item resolves at execution or organization time.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Validate workspace-level ownership | Fabric portal > Workspaces > target workspace > Workspace settings | Setting category matches the repeated symptom across items |

| Confirm Spark default behavior | Fabric portal > Workspace settings > Spark settings, then run representative notebook | Notebook starts with the expected Spark configuration |
| Confirm domain assignment | Fabric portal > Workspace settings > Domain | Workspace appears under the intended domain context |

| Confirm Dataflows Gen2 behavior | Fabric portal > workspace Dataflow Gen2 item > refresh history | Refresh uses the expected workspace-supported behavior |

Implement lifecycle management with Git integration, database projects, and deployment pipelines

Exam Radar

- Core Priority: Lifecycle management controls how Fabric engineering assets move from development to test and production without uncontrolled manual edits.
- High Frequency: DP-700 scenarios ask whether to use version control, database projects, or deployment pipelines when a team needs repeatable release promotion.
- Confusion Alert: Git integration tracks source changes; deployment pipelines promote Fabric content between stages; database projects support database object development. They are related but not interchangeable.
- Scenario Logic: If the requirement is code history and pull-request review, choose version control. If the requirement is stage promotion, choose deployment pipelines. If the requirement is database schema source management, choose database projects.
- Version Delta: The current guide explicitly includes configuring version control, implementing database projects, and creating deployment pipelines.
- Failure Trigger: Manual production edits, unreviewed schema drift, or promotion from the wrong stage.
- Operational Dependency: Workspace items must be supported by the lifecycle feature, and the user must have permissions in both Fabric and the linked source-control or pipeline stage.
- How the Exam Asks It: The question gives a release-governance objective and asks which Fabric lifecycle control satisfies it.
- How Distractors Are Designed: Distractors often use refresh schedules, sensitivity labels, or workspace roles even though the issue is change movement.
- Why the Correct Answer Works: The correct feature owns the change-control boundary named in the requirement.

Atomic Deconstruction - Operational Level

Lifecycle management starts by separating authoring, review, and release. Git integration records item definitions and enables review workflows outside the live production workspace. Deployment pipelines provide a Fabric-native promotion path between stages. Database projects let warehouse or SQL database schema work be treated as a source-managed engineering asset rather than an undocumented production edit.

The why-layer is control of state. A production workspace should not be the first place a change exists. A Git branch or database project captures the intended definition. A deployment pipeline moves a tested definition into a higher stage. Without this sequence, production may contain a mixture of manual changes, stale dependencies, and untraceable object versions.

Component Specifications

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

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

| Git integration | Connected branch and folder | Supported provider and branch path | Not connected | Repository access and workspace permissions | Changes cannot be reviewed or traced |

| Deployment pipeline | Stage mapping | Development, test, production-style stages | No pipeline | Supported item types and stage workspace access | Wrong artifact version is promoted |

| Database project | Schema object definition | Tables, views, procedures, security objects supported by project tooling | External to Fabric item until configured | SQL development workflow and source repository | Schema drift or missing dependency appears during deployment |

| Fabric item | Deployment status | Same, different, missing, unsupported | Uncompared | Pipeline comparison metadata | Promotion skips required dependency or overwrites unexpected changes |

Step-by-Step Execution Path

1. Identify the release boundary: source review, schema management, or stage promotion. This prevents choosing a lifecycle feature by product familiarity instead of requirement ownership.
2. Configure Git integration for the development workspace when code review and history are required. Use the Fabric portal path for workspace Git integration and validate provider, branch, and folder selection.
3. Put database object definitions into a database project when the requirement names warehouse or database schema lifecycle. Validate that object dependencies are included before promotion.
4. Create a deployment pipeline and assign workspaces to stages when the requirement is controlled movement from development to test or production.
5. Compare stages before deployment. The comparison is the checkpoint that exposes missing, changed, or unsupported items.
6. Deploy only after dependencies are visible and expected. Use deployment history or item comparison status as evidence.

Supported verification should come from Fabric portal Git status, deployment pipeline comparison, deployment history, source-control review records, and database project build or validation output for the active tooling version.

Technical Chain

A developer changes an item definition in the development workspace. Git integration serializes supported item metadata to a repository branch, where review and history occur. When approved, the Fabric deployment pipeline compares the source stage and target stage. The pipeline then applies supported item changes into the target workspace. If a database object is managed through a project, the schema definition is validated

before it becomes target state. Incorrect lifecycle selection breaks the chain: Git alone does not promote stages, and a deployment pipeline alone does not provide branch review semantics.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Validate Git connection | Fabric portal > Workspace settings > Git integration | Repository, branch, and sync status are visible |

| Validate stage differences | Fabric portal > Deployment pipelines > Compare | Expected changed, new, or missing items are listed |
| Validate promotion history | Fabric portal > Deployment pipelines > Deployment history | Target stage shows successful deployment for expected items |

| Validate database project readiness | Supported database project build or validation output | Schema objects compile and dependency errors are absent |

Configure security and governance across workspace, item, row, column, object, file, labels, endorsement, and audit logs

Exam Radar

- Core Priority: Security in Fabric is layered. The exam tests whether the candidate chooses the lowest effective control for the protected object and evidence requirement.
- High Frequency: Questions often combine workspace access, item sharing, row-level security, column-level security, object-level security, folder or file permissions, dynamic data masking, sensitivity labels, endorsement, and audit logs.
- Confusion Alert: Workspace role assignment is not the same as row-level security. Sensitivity labels classify and protect data but do not replace access control. Endorsement signals trust but does not grant permission.
- Scenario Logic: Start with the protected boundary. Use workspace access for workspace membership, item access for individual artifacts, row or column controls for query result shaping, labels for classification, endorsement for trust, and audit logs for investigation.
- Version Delta: The current guide includes Fabric audit logs and multiple fine-grained control levels.
- Failure Trigger: Overbroad workspace roles, missing item permission, visible sensitive columns, masked values expected but not applied, or no audit evidence for access investigation.
- Operational Dependency: Microsoft Purview, tenant settings, item support, and identity configuration may affect which governance features are available.

- How the Exam Asks It: The stem names a security outcome and asks for the Fabric feature that enforces or proves it.
- How Distractors Are Designed: Wrong answers frequently use classification for access enforcement or endorsement for data protection.
- Why the Correct Answer Works: The correct answer maps to the enforcement or evidence layer that owns the named behavior.

Atomic Deconstruction - Operational Level

Fabric security decisions begin with scope. Workspace roles define broad authoring and consumption boundaries. Item-level access grants access to a specific lakehouse, warehouse, semantic model, notebook, pipeline, or other item. Row-level, column-level, object-level, and folder/file-level controls narrow what an authorized user can see or query. Dynamic data masking changes visible values for supported objects without redesigning the table. Sensitivity labels apply classification and protection metadata. Endorsement tells users which items are promoted or certified. Audit logs provide evidence of access and administrative activity.

The dependency is identity-to-object evaluation. A user first needs an identity and a permission path. Then the query or item operation evaluates fine-grained rules. Labels and endorsement add governance signals, while audit logs record activity. If the wrong layer is chosen, the user either receives too much access, no access, or only a label with no enforcement.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Workspace role | Member capability | Admin, Member, Contributor, Viewer-style roles | No role unless assigned | Microsoft Entra identity and workspace permission | User cannot access workspace or receives excessive authoring rights |

| Item permission | Artifact access | Read, build, share, edit-style permissions by item | Not shared | Item owner or workspace role | User can access workspace but not target item |
| Row-level security | Predicate logic | User or group filtered rows | Not applied | Supported engine and identity mapping | User sees rows outside allowed segment |

| Dynamic data masking | Masking rule | Supported mask formats by data store | Unmasked | Supported warehouse or SQL object feature | Sensitive values appear in query output |
| Audit log | Activity event | User, operation, item, timestamp | Captured when auditing is enabled and available | Microsoft Purview audit capability and permissions | Investigation lacks authoritative event evidence |

Step-by-Step Execution Path

1. Classify the requirement as access, data-shaping, classification, trust, or evidence. This prevents applying labels when the requirement demands enforcement.

2. Check workspace membership and role. If the user cannot reach the workspace, fine-grained controls never execute.
3. Check item permission. If only one artifact is required, item sharing can avoid overbroad workspace role assignment.
4. Apply row, column, object, folder, file, or masking controls when the user should access the item but not all data inside it.
5. Apply sensitivity labels when the question asks for classification, protection metadata, or compliance marking.
6. Use endorsement when the question asks users to identify trusted content.
7. Use audit logs when the question asks who accessed or changed an item.

Evidence should come from Fabric permission dialogs, supported SQL security definitions, item access views, Microsoft Purview sensitivity label state, endorsement badge state, and Microsoft Purview audit search results.

Technical Chain

When a user opens or queries a Fabric item, Fabric evaluates workspace membership and item permission before the engine returns data. If the engine supports row, column, object, masking, folder, or file controls, those rules shape the returned result or accessible object set. Sensitivity labels and endorsement are evaluated as governance metadata and user signals, not as substitutes for permission checks. Audit events are emitted from the service activity path so an investigator can reconstruct who performed the operation and when.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Validate workspace role | Fabric portal > Workspace > Manage access | User or group has the intended role only |

| Validate item access | Fabric portal > Item > Manage permissions | Target identity has only required item permission |
| Validate row or column behavior | Run supported query as test identity | Result contains only allowed rows or protected columns |

| Validate audit evidence | Microsoft Purview audit search for Fabric activity | Event includes user, item, operation, and timestamp |

Orchestrate processes with Dataflows Gen2, pipelines, notebooks, schedules, triggers, parameters, and dynamic expressions

Exam Radar

- Core Priority: Orchestration questions test whether the candidate can select the right Fabric execution vehicle and connect activities with parameters, schedules, triggers, and dynamic expressions.
- High Frequency: Expect comparisons between Dataflows Gen2, pipelines, and notebooks for transformation, movement, and code-driven processing.
- Confusion Alert: Dataflows Gen2 is not a general workflow orchestrator. Pipelines coordinate activities. Notebooks run code-heavy Spark or data engineering logic.
- Scenario Logic: Choose Dataflows Gen2 for low-code data preparation, pipelines for orchestration and activity coordination, and notebooks for code-driven Spark transformations or custom logic.
- Version Delta: The current guide explicitly includes schedules, event-based triggers, notebooks, pipelines, parameters, and dynamic expressions.
- Failure Trigger: Hardcoded paths, wrong trigger type, missing parameter binding, or a notebook that runs manually but fails inside a pipeline.
- Operational Dependency: Pipeline activities must pass parameters in the format expected by the target notebook, dataflow, or activity. Trigger identity and schedule state must be enabled.
- How the Exam Asks It: The stem describes an automated data process and asks which artifact or configuration makes it repeatable.
- How Distractors Are Designed: Wrong answers choose a transformation tool when the need is orchestration, or choose a schedule when the requirement is event-based.
- Why the Correct Answer Works: The correct answer owns the process-control requirement rather than only the transformation logic.

Atomic Deconstruction - Operational Level

A Fabric orchestration design has three decisions: execution vehicle, trigger, and runtime parameterization. Dataflows Gen2 handles many low-code ingestion and shaping tasks. Pipelines coordinate activities, dependencies, triggers, and failure paths. Notebooks handle programmable Spark logic, libraries, and custom validation. Parameters remove hardcoded environment and date values; dynamic expressions bind runtime values to activity inputs.

The why-layer is repeatability. A process that depends on manual date edits or manual notebook execution is not operationally stable. A schedule or event trigger starts the process. A pipeline passes parameter values. Dependency conditions prevent downstream writes after validation failure. Without these controls, a technically correct transformation can run at the wrong time, against the wrong partition, or after a failed upstream step.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Pipeline | Activity graph | Copy, notebook, dataflow, script, validation-style activities | Empty until authored | Data Factory workload and item permissions | Activities run out of order or not at all |

| Trigger | Schedule or event condition | Time-based or supported event-based trigger | Disabled or not configured | Workspace and trigger permissions | Process requires manual start |
| Notebook activity | Parameter binding | Name-value parameter map | No runtime value | Notebook parameter handling | Notebook runs with wrong date, path, or environment |

| Dynamic expression | Runtime value resolution | Supported expression language | Literal value | Activity metadata and parameter scope | Hardcoded or invalid expression breaks run |
| Dependency condition | Success, failure, completion, skipped paths | Activity outcome states | No explicit branch | Upstream activity output | Invalid data is written after failed validation |

Step-by-Step Execution Path

1. Decide whether the primary need is transformation or coordination. If the requirement includes branching, scheduling, or multiple activities, start with a pipeline.
2. Add the ingestion, notebook, dataflow, validation, and write activities in dependency order. This ensures upstream state exists before downstream execution.
3. Define pipeline parameters such as processing date, source path, target environment, or batch identifier.
4. Bind parameters to notebook or activity inputs with dynamic expressions. Validate spelling and scope because a wrong parameter name can produce a runtime failure.
5. Configure success and failure paths. Put quality checks before target writes so invalid data does not become committed output.
6. Configure a schedule or event-based trigger based on the requirement wording. Enable it only after a manual test run succeeds.
7. Inspect pipeline run history and activity outputs after execution.

Use Fabric pipeline run history, activity input/output panes, notebook run status, Dataflow Gen2 refresh history, and trigger state as evidence.

Technical Chain

A trigger creates a pipeline run. The pipeline runtime resolves parameters and dynamic expressions, then schedules activities according to dependency conditions. When the notebook activity starts, it receives runtime values instead of hardcoded constants. Validation output determines whether downstream activities

execute. A wrong parameter binding breaks the chain at activity start; a missing dependency condition lets downstream writes run even when upstream validation failed.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Validate trigger state | Fabric portal > Pipeline > Triggers | Trigger is enabled and matches required schedule or event |

| Validate parameter binding | Fabric portal > Pipeline activity > Settings > Parameters | Parameter names match the called item requirements |
| Validate dependency path | Fabric portal > Pipeline canvas dependency conditions | Downstream activity runs only on intended status |

| Validate execution evidence | Fabric portal > Pipeline > Run history > Activity output | Each activity shows expected input, output, and status |

Practice Questions

1. A Fabric team uses several notebooks in the same workspace. Each notebook should use the same Spark runtime defaults, but engineers keep changing session settings manually. What should the data engineer configure first?
   A. The Spark workspace settings for the Fabric workspace
   B. A row-level security rule on the lakehouse table
   C. A deployment pipeline between development and production
   D. A sensitivity label on every notebook

2. A development workspace must support pull request review of Fabric item changes and then promote approved changes to a test workspace. Which combination best matches the requirement?
   A. Microsoft Purview audit logs and sensitivity labels
   B. Git integration and deployment pipelines
   C. Dynamic data masking and object-level security
   D. OneLake shortcuts and mirrored databases

3. A user can open a Fabric warehouse item but should only see rows for the region they manage. Which control should be prioritized?
   A. Assign the user to Workspace Admin
   B. Endorse the warehouse as certified
   C. Configure row-level security for the regional filter
   D. Increase the Fabric capacity size

4. A pipeline must run every morning and pass the business date into a notebook activity so the notebook loads only the correct partition. What should the data engineer use?
   A. A sensitivity label and endorsement
   B. A deployment pipeline comparison
   C. A workspace domain assignment
   D. A schedule trigger with pipeline parameters and dynamic expressions

5. A workspace is assigned to the wrong data domain, and several items are appearing outside the intended governance grouping. What is the lowest-scope configuration to inspect?
   A. Notebook cell logic
   B. Warehouse query predicates
   C. Pipeline retry settings
   D. Workspace domain settings

6. A production workspace contains manual edits that do not match development. The team needs to compare stages before promoting changes. Which Fabric feature provides the most relevant evidence?
   A. Dataflow Gen2 refresh history
   B. Deployment pipeline comparison
   C. Lakehouse table optimization
   D. Dynamic data masking

7. A report consumer should see a warehouse table but should not see one sensitive column. Which security approach is most aligned?
   A. Add the consumer to Workspace Contributor
   B. Configure column-level security or an equivalent supported column protection control
   C. Create a new Eventstream for the table
   D. Increase the Spark executor memory

8. A Fabric process has three activities: ingest data, transform it, and update a quality table. The transform activity must wait until ingestion succeeds, and the quality table must receive a runtime load identifier. What design should be selected?
   A. A workspace role assignment for all users
   B. A sensitivity label applied to the pipeline
   C. A lakehouse shortcut to the source folder only
   D. A pipeline dependency chain with parameters passed between activities

9. A team wants warehouse schema definitions to be tracked and validated as engineering assets instead of being changed manually in production. What should they use?
   A. A Dataflow Gen2 staging option
   B. A workspace domain
   C. A database project
   D. A capacity metric view

10. A Fabric administrator needs evidence of who deleted an item last week. Which evidence source should be used?
    A. Spark workspace settings
    B. Lakehouse table file statistics
    C. Pipeline activity retry count
    D. Microsoft Purview audit logs for Fabric activity

Ingest and transform data

Design and implement full, incremental, dimensional, and streaming loading patterns

Exam Radar

- Core Priority: Loading pattern selection determines cost, latency, correctness, and recoverability before the tool choice is made.
- High Frequency: DP-700 frequently asks candidates to choose between full load, incremental load, dimensional preparation, and streaming ingestion patterns.
- Confusion Alert: Incremental loading is not simply a smaller full load. It requires a reliable change signal, watermark, timestamp, version, event offset, or source change mechanism.
- Scenario Logic: Use full loads when volume is small or replacement is required; incremental loads when changes can be detected; dimensional preparation when facts and dimensions must be conformed; streaming when low-latency event processing is required.
- Version Delta: The current guide includes full and incremental data loads, dimensional model preparation, and streaming load pattern design.
- Failure Trigger: Duplicate facts, missed late-arriving rows, dimension key mismatch, unbounded stream backlog, or overwrite of historical data.
- Operational Dependency: The source must provide a trustworthy change indicator or event ordering signal for incremental and streaming designs.
- How the Exam Asks It: The stem gives source behavior and target modeling requirements, then asks for the best load pattern.
- How Distractors Are Designed: Distractors often choose a faster tool without satisfying change detection, keys, or latency requirements.
- Why the Correct Answer Works: The correct pattern satisfies the data-state dependency first, then can be implemented with a suitable Fabric tool.

Atomic Deconstruction - Operational Level

A load pattern is a contract between source change behavior and target state. Full load replaces or rebuilds the target from the complete source. Incremental load reads only data beyond a stored watermark or change boundary. Dimensional loading prepares facts, dimensions, surrogate keys, slowly changing attributes, and conformed structures for analytical models. Streaming loading processes events continuously or micro-batches them with ordering and checkpoint logic.

The why-layer is correctness under change. A full load can be correct but expensive. Incremental loading is efficient but only if the watermark cannot miss updates. Dimensional preparation prevents analytical joins from using unstable operational keys. Streaming ingestion is necessary when the business question depends on event freshness and backlog control.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Watermark | Change boundary | Timestamp, monotonically increasing ID, version, offset | Not stored | Reliable source change column or event offset | Duplicate or missed changed rows |

| Fact table | Grain | Transaction, snapshot, accumulating snapshot | Undefined until modeled | Stable business process definition | Measures double count or join incorrectly |
| Dimension table | Key strategy | Natural key, surrogate key, SCD handling | Source key only | Conformed modeling rules | Historical attributes overwrite incorrectly |

| Streaming source | Event ordering and checkpoint | Event time, processing time, offset, checkpoint | No checkpoint | Eventstream or supported streaming source | Backlog grows or events are reprocessed |
| Target Delta table | Merge behavior | Append, overwrite, upsert, merge | Depends on operation | Table schema and key match | Incorrect overwrite or duplicate rows |

Step-by-Step Execution Path

1. Inspect source volume, latency requirement, change signal, and target model. Pattern selection happens before selecting a Fabric item.
2. For full load, validate that replacement windows, downstream dependencies, and target rebuild time are acceptable.
3. For incremental load, define the watermark and persist it outside the transient run state. The next run must know what the previous run processed.
4. For dimensional loading, define fact grain, dimension keys, late-arriving data handling, and attribute-change behavior before writing tables.
5. For streaming, validate source event ordering, checkpoint behavior, and target latency requirement.
6. Reconcile source counts, target counts, watermark values, and exception rows after each run.

Verification can use source query counts, target Delta table history, pipeline run output, notebook validation cells, warehouse reconciliation queries, and streaming backlog or checkpoint evidence.

Technical Chain

The source emits either a full data state, a detectable change set, or an event sequence. The load design chooses how to transform that source state into target state. A persisted watermark causes the next run to request only changes beyond the last successful boundary. Dimensional rules convert operational rows into analytics-ready facts and dimensions. Streaming checkpoints record progress through an event sequence. If the boundary is wrong, the target is not merely slow; it becomes logically incorrect.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Validate watermark continuity | Query control table or run metadata for last successful watermark | Current run starts after previous successful boundary |

| Validate target merge result | Supported SQL or Spark count comparison against expected changed keys | Inserted and updated row counts match source change set |
| Validate dimensional grain | Warehouse or lakehouse SQL query grouping by candidate key | No unintended duplicate fact rows at declared grain |

| Validate streaming progress | Eventstream or notebook checkpoint/backlog view | Offsets or backlog show forward progress without replay storm |

Ingest and transform batch data with Lakehouse, Warehouse, Dataflows Gen2, notebooks, KQL, T-SQL, shortcuts, mirroring, and pipelines

Exam Radar

- Core Priority: Batch ingestion questions test matching the data store and transformation engine to source format, engineering skill, target serving pattern, and governance requirements.
- High Frequency: Expect choices between Lakehouse, Warehouse, Dataflows Gen2, notebooks, KQL, T-SQL, pipelines, shortcuts, and mirroring.
- Confusion Alert: OneLake shortcuts reference data; they do not copy and transform it by themselves. Mirroring replicates supported source data into Fabric; it is not a generic notebook transformation.
- Scenario Logic: Lakehouse fits open data engineering and Spark/Delta workflows. Warehouse fits relational SQL analytics. Dataflows Gen2 fits low-code transformations. Notebooks fit code-heavy transformation. KQL fits real-time analytics and Eventhouse-style query patterns. T-SQL fits warehouse transformations.

- Version Delta: The current guide includes choosing data stores and transformation tools, managing shortcuts, implementing mirroring, using pipelines, and resolving shortcut errors.
- Failure Trigger: Wrong engine for transformation language, shortcut authorization failure, unsupported mirrored source, or pipeline copy without schema reconciliation.
- Operational Dependency: Source connectivity, credentials, OneLake permissions, supported mirroring source, and target schema must be validated before transformation.
- How the Exam Asks It: The stem gives source, transformation language, and target consumption needs.
- How Distractors Are Designed: Wrong answers select a familiar tool that cannot satisfy language, latency, storage, or support constraints.
- Why the Correct Answer Works: The correct option aligns data store, transformation engine, and operational dependency.

Atomic Deconstruction - Operational Level

Batch ingestion in Fabric separates source connection, movement or reference, transformation engine, and target serving. Pipelines move and orchestrate data. Dataflows Gen2 provide low-code preparation. Notebooks execute PySpark or custom code. Warehouses use T-SQL for relational transformation. KQL is used where event or real-time analytics query patterns apply. Shortcuts expose external or internal data through OneLake without copying the data. Mirroring keeps supported operational data synchronized into Fabric.

The why-layer is tool fitness. A shortcut failure is usually a permission, path, or external location issue, not a Spark optimization problem. A T-SQL warehouse transformation should not be forced into KQL unless the target is an Eventhouse analytics pattern. Mirroring only works when the source is supported and configuration prerequisites are met.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Lakehouse | Table format | Delta tables, files, shortcuts | Empty until loaded | OneLake and Spark permissions | Files exist but tables are not queryable as expected |

| Warehouse | Transformation language | T-SQL | Empty schema | Warehouse item permissions | SQL consumers cannot access curated relational model |
| Dataflow Gen2 | Transformation mode | Low-code Power Query-style steps | No refresh | Connector credentials and destination | Refresh fails or schema drifts |

| OneLake shortcut | Target path and credential | Internal or supported external location | Not created | Source permissions and shortcut support | Shortcut error, inaccessible files, broken path |

| Mirroring | Source replication configuration | Supported mirrored source types | Not configured | Source compatibility and permissions | Replication lag or unsupported source error |

Step-by-Step Execution Path

1. Identify source type, target serving pattern, and transformation language. This determines the Fabric store and engine.
2. Choose Lakehouse for Spark/Delta engineering, Warehouse for relational SQL serving, Dataflows Gen2 for low-code preparation, notebook for PySpark/custom logic, KQL for real-time analytics patterns, and pipeline for movement or orchestration.
3. If using shortcuts, validate source path, credential, permission, and supported location before troubleshooting downstream transformations.
4. If using mirroring, confirm the source is supported and replication status is healthy before querying target data.
5. Use a pipeline when ingestion must be scheduled, parameterized, retried, or combined with other activities.
6. Validate schema, row counts, error rows, refresh status, and target accessibility after ingestion.

Evidence should come from Fabric item run history, Dataflow Gen2 refresh details, pipeline activity output, Lakehouse table preview, Warehouse SQL query output, shortcut status, and mirroring replication status.

Technical Chain

The batch source is connected by credential or referenced by shortcut. A pipeline, dataflow, notebook, SQL script, or KQL query then transforms or moves data into a serving store. OneLake stores or references the data path. The target engine reads the data according to table metadata and permissions. If a shortcut credential fails, the engine never receives readable files. If the wrong transformation engine is chosen, the logic may be unsupported even though the data is present.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Validate shortcut health | Fabric portal > Lakehouse > Shortcuts | Shortcut resolves and files are browseable |

| Validate Dataflow Gen2 refresh | Fabric portal > Dataflow Gen2 > Refresh history | Latest refresh succeeds with expected row counts |
| Validate pipeline ingestion | Fabric portal > Pipeline > Run history > Copy activity output | Read/write counts and error rows match expectation |

| Validate warehouse serving | Warehouse SQL query against curated table | Schema and row counts match the batch load contract |

Transform data with SQL, PySpark, and KQL while preserving schema, quality, and analytical intent

Exam Radar

- Core Priority: DP-700 expects candidates to manipulate and transform data with SQL, PySpark, and KQL. The exam tests when each language is operationally appropriate.
- High Frequency: Scenarios compare code-heavy transformation, relational warehouse shaping, and real-time analytics querying.
- Confusion Alert: SQL, PySpark, and KQL can all filter or aggregate data, but their execution contexts and target stores differ.
- Scenario Logic: Use PySpark for distributed file and Delta transformations, SQL for relational warehouse modeling and query serving, and KQL for high-performance event or log analytics patterns.
- Version Delta: The official audience profile names SQL, PySpark, and KQL skills.
- Failure Trigger: Schema mismatch, wrong data type conversion, skipped null handling, non-idempotent transformation, or language used outside its supported item context.
- Operational Dependency: Transformation logic must match the Fabric item, table schema, security model, and refresh or run orchestration.
- How the Exam Asks It: A stem describes a transformation objective and a Fabric context, then asks which language or pattern is best.
- How Distractors Are Designed: Distractors select a language because it can express the calculation, while ignoring the item context or serving target.
- Why the Correct Answer Works: The correct option fits both transformation semantics and execution location.

Atomic Deconstruction - Operational Level

Transformation design includes input schema, data quality rule, transformation language, output schema, and validation query. SQL is strong for relational joins, dimensional shaping, constraints, and warehouse serving. PySpark is strong for scalable file processing, Delta operations, and programmable transformations. KQL is strong for time-series, event, telemetry, and log-style analytics in supported real-time analytics contexts.

The why-layer is preserving analytical intent. A transformation is not correct just because it returns rows. Data types, nulls, keys, timestamps, partition columns, and aggregation grain must match the intended analytical question. If a PySpark job writes strings for numeric measures, downstream SQL reports may silently misaggregate. If a KQL query uses processing time instead of event time, monitoring conclusions can be wrong.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| SQL transformation | Join and aggregation grain | T-SQL supported expressions | Query-only until materialized | Warehouse or SQL endpoint | Duplicate joins or incorrect measures |

| PySpark transformation | DataFrame schema and Delta write mode | append, overwrite, merge-style logic | In-memory until written | Spark session and OneLake permissions | Schema drift or duplicate output |
| KQL query | Time filter and summarize clause | Supported KQL operators | Query-only | Eventhouse or KQL database context | Wrong time window or high query cost |

| Quality rule | Null, range, uniqueness, referential check | Pass, warn, fail | Not enforced unless coded | Source schema and business rule | Bad rows enter curated layer |
| Output table | Schema contract | Column names, types, partitions | Created by first write or DDL | Target engine and permissions | Downstream model breaks on type or name change |

Step-by-Step Execution Path

1. Read the target Fabric item and serving requirement before writing transformation logic. The item context narrows the language choices.
2. Profile source schema, nulls, keys, and timestamp fields. This establishes quality checks before transformation.
3. Choose SQL for warehouse relational transformations, PySpark for scalable lakehouse transformations, or KQL for event/log analytics queries.
4. Implement transformation with explicit output schema, key behavior, and error handling. Avoid implicit type conversion where downstream analytics depend on precision.
5. Validate output row counts, duplicate keys, null exceptions, and business-rule totals.
6. Attach the transformation to a pipeline or schedule only after the transformation passes validation on representative data.

Use SQL query output, notebook cell output, KQL query results, table schema inspection, run history, and data-quality exception tables as verification evidence.

Technical Chain

The engine reads source data according to its metadata and execution model. SQL optimizes relational operations against warehouse structures. PySpark distributes transformations across Spark executors and writes table files and metadata. KQL evaluates time-window and event analytics operators against the KQL store. The target schema becomes the contract for downstream consumers. If the selected engine does not match the storage or query pattern, the transformation can become slow, unsupported, or semantically wrong.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Validate SQL output schema | Warehouse query: inspect columns and data types with supported metadata views | Column names and types match target contract |

| Validate PySpark write result | Notebook displays Delta table schema and count after write | Schema and row count match expected transformation output |
| Validate KQL time-window logic | KQL query with explicit time filter and summarize output | Returned events fall inside required window and aggregation grain |

| Validate quality exceptions | Query exception table or notebook validation output | Bad rows are isolated and not silently loaded |

Practice Questions

1. A source system provides a reliable modified timestamp, and the Fabric lakehouse should load only records changed since the previous successful run. Which loading pattern is most appropriate?
   A. Full reload of every source table on every run
   B. Random sampling of source records
   C. Manual copy of files into OneLake
   D. Incremental loading based on a watermark

2. A batch ingestion pipeline writes transformed data to a lakehouse, but downstream users need slowly changing dimension logic for customer attributes. What should the engineer design?
   A. A sensitivity label for the customer table
   B. A dimensional loading pattern that handles current and historical attribute state
   C. A workspace domain assignment only
   D. A capacity resize before defining the model

3. A team needs to ingest data from an external storage location without copying every file into a new managed folder first. Which Fabric object is most aligned?
   A. Dynamic data masking
   B. A deployment pipeline stage
   C. A OneLake shortcut
   D. A Power BI endorsement label

4. A Dataflow Gen2 refresh fails at the connector step before any transformation logic runs. What should the data engineer inspect first?
   A. Warehouse query plan shape
   B. Source connection, credentials, and connector error details

C. Spark shuffle partition skew
D. Deployment pipeline comparison status

5. A transformation must be written in SQL because the target is a warehouse and the team needs set-based joins and aggregations. Which approach fits best?
   A. Create only an Eventstream
   B. Change the workspace domain
   C. Apply a sensitivity label to the workspace
   D. Write T-SQL transformations against the warehouse objects

6. A notebook transformation fails after a source column changes type. The destination table expects the original schema. Which action should be prioritized?
   A. Increase the capacity size immediately
   B. Endorse the notebook
   C. Validate schema drift and update the transformation or target schema intentionally
   D. Add a deployment pipeline stage

7. A near-real-time scenario requires event data to be captured continuously and made available for analysis. Which pattern is most appropriate?
   A. A monthly full reload into a warehouse only
   B. A manual export from the Fabric portal
   C. A row-level security predicate
   D. A streaming ingestion pattern with Eventstreams or Eventhouse-related processing where supported

8. A mirrored database is being used so Fabric can analyze operational data with minimal custom pipeline movement. What is the key reason to choose mirroring?
   A. It replaces all security controls in the workspace
   B. It supports near-current replicated access to supported source data without building the same copy pipeline manually
   C. It automatically certifies every downstream item
   D. It removes the need to validate schema and permissions

9. A PySpark job produces duplicate rows because the merge condition does not uniquely identify target records. What should the engineer inspect first?
   A. The merge key and deduplication logic used by the transformation
   B. The workspace endorsement status
   C. The audit log retention period
   D. The deployment pipeline name

10. A KQL query over event data returns too many rows and performs poorly because it scans old data that the scenario does not need. What should the engineer adjust first?
    A. Add a sensitivity label
    B. Create a database project

C. Apply appropriate time filters and projections in the KQL query

D. Promote the workspace to production

Monitor and optimize an analytics solution

Identify and resolve ingestion, transformation, shortcut, permission, refresh, and orchestration errors

Exam Radar

- Core Priority: Monitoring questions test the first diagnostic signal and the control object that owns the failure.
- High Frequency: DP-700 scenarios include failed pipeline runs, Dataflow Gen2 refresh errors, notebook failures, shortcut resolution errors, permission denials, and query failures.
- Confusion Alert: Retrying is not diagnosis. Scaling capacity is not the first fix when the error says authentication, path not found, schema mismatch, or missing permission.
- Scenario Logic: Read the error location first, map it to the owning object, then inspect the dependency that object requires.
- Version Delta: The current guide includes identifying and resolving errors as part of monitoring and optimizing an analytics solution.
- Failure Trigger: Expired credential, renamed path, schema drift, missing workspace or item permission, broken shortcut target, invalid parameter, or failed upstream activity.
- Operational Dependency: Run history, refresh history, activity output, logs, metrics, and permission views must be available to the operator.
- How the Exam Asks It: The stem provides a symptom and asks for the first action or most likely cause.
- How Distractors Are Designed: Wrong answers jump to optimization, rebuilds, or unrelated governance controls before reading the failure evidence.
- Why the Correct Answer Works: The correct answer uses the nearest authoritative error signal and fixes the failed dependency.

Atomic Deconstruction - Operational Level

Error resolution begins with run evidence. Pipeline run history identifies the failed activity and output payload. Dataflow refresh history shows connector, schema, transformation, or destination errors. Notebook output shows cell-level exceptions and Spark behavior. Shortcut status reveals path, credential, or external location issues. Permission views show whether the identity can access the workspace, item, or data object.

The why-layer is dependency localization. Every Fabric failure has an owner: trigger, activity, notebook, dataflow, shortcut, credential, schema, permission, or engine. Fixing the wrong owner wastes time and may

introduce risk. A schema drift error needs schema handling, not capacity scaling. A shortcut credential error needs path or permission correction, not notebook code tuning.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Pipeline run | Activity status and output | Succeeded, failed, canceled, skipped | No run until triggered | Activity configuration and credentials | Failed activity blocks downstream process |

| Dataflow Gen2 refresh | Step and connector error | Refresh success or failure details | No refresh evidence until run | Source credential and destination | Refresh stops at connector or transformation step |
| Notebook run | Cell exception and Spark state | Completed, failed, canceled | Manual or pipeline run context | Spark runtime, parameters, data access | Cell fails or writes incomplete output |

| OneLake shortcut | Resolution and credential state | Healthy or error state | Not validated until accessed | Target path and permission | Files cannot be browsed or read |
| Permission assignment | Identity and scope | Workspace, item, data object | No access unless assigned | Microsoft Entra identity | 401, 403, hidden item, or query denial |

Step-by-Step Execution Path

1. Locate the failing run or operation. Use pipeline run history, refresh history, notebook output, shortcut status, or query error text.
2. Identify the exact boundary where failure appears: trigger start, activity invocation, source read, transformation, target write, shortcut resolution, or permission evaluation.
3. Inspect the owner object at that boundary. For pipeline-to-notebook failures, check activity parameters and notebook expected inputs before editing notebook logic.
4. Validate credentials and permissions using the same identity or service context as the failed run.
5. Check schema and path dependencies only after identity and connection state are confirmed.
6. Re-run the smallest failing unit and compare status, output rows, and error details.

Use Fabric run history, refresh logs, notebook output, shortcut browse behavior, item permission panels, SQL/KQL error messages, and capacity metrics as evidence.

Technical Chain

A scheduled or manual operation creates a service execution context. That context resolves identity, item configuration, parameters, source path, schema, and target write permissions. If any dependency fails, the owning runtime returns an error at the nearest observable boundary. Reading that boundary first prevents symptom-only remediation. A retry without fixing credential, path, schema, or parameter state simply repeats the same dependency failure.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Validate failed activity evidence | Fabric portal > Pipeline > Run history > Failed activity output | Error message identifies activity and dependency |

| Validate Dataflow error step | Fabric portal > Dataflow Gen2 > Refresh history > Details | Failed connector, step, or destination is visible |
| Validate notebook exception | Notebook run output or pipeline notebook activity output | Failing cell and parameter context are visible |

| Validate shortcut access | Fabric portal > Lakehouse > Shortcuts > Browse target | Shortcut opens expected path without credential error |

Optimize Lakehouse tables, pipelines, warehouses, Eventstreams, Eventhouses, Spark jobs, and queries

Exam Radar

- Core Priority: Optimization requires matching the bottleneck to the object: table layout, pipeline activity, warehouse query, event stream, Eventhouse query, Spark job, or capacity.
- High Frequency: DP-700 asks which object to optimize when performance is slow or resource use is high.
- Confusion Alert: Capacity scaling is not always the first answer. Table layout, query shape, partition strategy, activity parallelism, or Spark configuration may be the controlling issue.
- Scenario Logic: Inspect metrics and execution evidence before changing configuration. Optimize the narrowest object that owns the bottleneck.
- Version Delta: The current guide includes optimizing lakehouse tables, pipelines, warehouses, Eventstreams and Eventhouses, Spark performance, and query performance.
- Failure Trigger: Small-file accumulation, unfiltered scans, skewed Spark partitions, inefficient pipeline activity order, warehouse query bottleneck, or event processing backlog.
- Operational Dependency: Performance evidence must identify whether the delay is storage scan, compute execution, orchestration wait, query plan, or streaming backlog.
- How the Exam Asks It: The stem gives a symptom such as slow query, long pipeline, Spark skew, or stream delay and asks for the best optimization action.
- How Distractors Are Designed: Distractors apply a generic performance feature without matching the bottleneck evidence.
- Why the Correct Answer Works: The correct action targets the resource or object producing the measured delay.

Atomic Deconstruction - Operational Level

Optimization starts with measurement. Lakehouse table optimization focuses on file size, table maintenance, partitioning, and scan reduction. Pipeline optimization focuses on activity order, parallelism, copy settings, dependency paths, and retry behavior. Warehouse optimization focuses on query shape, distribution of work, statistics or supported tuning features, and relational design. Eventstreams and Eventhouses focus on ingestion throughput, backlog, retention, and query efficiency. Spark optimization focuses on partitioning, shuffle, skew, caching, and runtime settings. Query optimization focuses on filters, joins, projections, and engine-specific execution plans.

The why-layer is bottleneck ownership. A slow query over a poorly maintained table will not be fixed by adding a pipeline retry. A Spark shuffle skew issue will not be fixed by a sensitivity label. Optimization must reduce the measured cost at the point where the system spends time, memory, or throughput.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Lakehouse Delta table | File layout and maintenance state | Small files, compacted files, partitioned data | Depends on writes | Table maintenance support and workload pattern | Slow scans and high metadata overhead |

| Pipeline | Activity concurrency and dependency graph | Sequential or parallel where supported | Authored by designer | Source and target throughput | Long wall-clock runtime or avoidable waits |
| Warehouse query | Predicate, join, aggregation, plan behavior | Engine-supported SQL patterns | Query text as submitted | Table design and statistics-like metadata where supported | Excessive scan or slow join |

| Spark job | Partitioning and shuffle behavior | Balanced or skewed partitions | Derived from data and code | Spark runtime and data distribution | Executor spill, skew, long stage runtime |
| Eventhouse or Eventstream | Ingestion and query throughput | Healthy, delayed, backlogged | Depends on source rate | Capacity and configuration | Late data, query latency, dropped or delayed processing |

Step-by-Step Execution Path

1. Capture performance evidence first: pipeline duration, activity output, query duration, Spark stage timing, table file pattern, or streaming backlog.
2. Classify the bottleneck as storage layout, orchestration, relational query, Spark execution, event ingestion, or query design.
3. For lakehouse tables, inspect file count, partition approach, and table maintenance options before changing compute.
4. For pipelines, inspect activity dependencies, parallel opportunities, copy throughput, and retry patterns.

5. For warehouses, inspect query predicates, joins, projections, and table design using supported query monitoring evidence.
6. For Spark jobs, inspect shuffle, skew, partition counts, and expensive transformations.
7. For Eventstreams and Eventhouses, inspect backlog, ingestion rate, retention, and query filters.
8. Re-measure the same workload after one targeted change.

Use supported Fabric monitoring views, run history, Spark UI or notebook metrics where exposed, warehouse query monitoring, Eventstream/Eventhouse monitoring, and table inspection evidence.

Technical Chain

The workload reads data, schedules compute, executes transformations or query operators, and writes or returns results. Each stage consumes time and resources. Small files increase metadata and scan overhead. Bad joins increase shuffle or relational work. Sequential pipeline dependencies increase wall-clock time. Streaming backlog grows when ingestion exceeds processing. The optimization changes the cost driver at the measured stage; if the change targets a different object, the original cost remains.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ---------- | ---------- | ---------- |
| Validate lakehouse table scan issue | Fabric Lakehouse table details or supported notebook table inspection | File count, partition layout, or scan pattern explains slow query |

| Validate pipeline bottleneck | Fabric portal > Pipeline > Run history > Activity durations | Longest activity or avoidable wait path is identified |
| Validate warehouse query behavior | Warehouse query monitoring or supported execution evidence | Slow query, scan, join, or wait pattern is visible |

| Validate Spark skew | Notebook Spark execution details or Spark UI where available | One or more stages or partitions dominate runtime |

Monitor analytics solutions with Fabric run history, metrics, audit evidence, and operational readiness criteria

Exam Radar

- Core Priority: Monitoring is the evidence layer that proves whether an analytics solution is healthy, secure, and ready for production operation.
- High Frequency: DP-700 scenarios ask what to inspect when a scheduled load is late, a refresh fails, a query slows, or an access event must be investigated.

- Confusion Alert: Audit logs answer who did what; metrics answer resource or performance behavior; run history answers execution status. They are not substitutes.
- Scenario Logic: Match evidence type to the question: run status for execution, metrics for capacity/performance, audit for user activity, logs or query output for detailed failure context.
- Version Delta: The current guide includes monitoring and optimizing analytics solutions, including audit logs under governance.
- Failure Trigger: No baseline, no alerting path, ignored refresh failures, missing audit permissions, or optimization without measurement.
- Operational Dependency: The operator must know where each evidence source lives and what question it can answer.
- How the Exam Asks It: The stem asks for the best evidence source or monitoring action for a named operational concern.
- How Distractors Are Designed: Distractors choose a configuration feature when the requirement is observation or investigation.
- Why the Correct Answer Works: The correct evidence source directly observes the operational state named in the stem.

Atomic Deconstruction - Operational Level

Monitoring a Fabric analytics solution requires evidence separation. Run history tells whether a pipeline, notebook, or dataflow ran and where it failed. Metrics show capacity pressure, throughput, latency, or resource patterns. Audit logs show user and administrative activity. Query and engine evidence show execution details. Readiness criteria combine these signals into operational standards such as successful scheduled runs, acceptable duration, controlled access, validated row counts, and known recovery steps.

The why-layer is operational confidence. A solution is not production-ready because it ran once manually. It must produce repeatable run evidence, expose failures, show acceptable performance, and provide auditability for sensitive operations. Without monitoring, failures become user-discovered incidents rather than controlled engineering events.

Component Specifications

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

| ---------- | ---------- | ---------- | ---------- | ---------- | ---------- |
| Run history | Status, duration, activity output | Succeeded, failed, canceled, skipped | Available after run | Pipeline, notebook, dataflow execution | Unknown failure point or missed SLA |

| Capacity or workload metric | Utilization, latency, throttling-style signal | Normal, elevated, saturated | Depends on workload | Monitoring access and capacity telemetry | Slow workloads without root evidence |

| Audit log | User, operation, item, timestamp | Searchable events where enabled | Requires audit capability | Microsoft Purview permissions and retention | Cannot prove who changed or accessed item |

| Data-quality checkpoint | Count, freshness, exception rows | Pass, warn, fail | Not present unless designed | Validation logic and target metadata | Bad data reaches consumers |
| Operational readiness criteria | SLA, recovery, validation, access review | Met or unmet | Undefined unless documented | Monitoring and ownership | Production handoff lacks measurable standard |

Step-by-Step Execution Path

1. Define the operational question: execution success, performance, access investigation, data quality, or readiness.
2. Inspect run history for scheduled data processes. Confirm status, duration, failed activity, and output details.
3. Inspect metrics when the question mentions slow performance, capacity pressure, throughput, or backlog.
4. Inspect audit logs when the question asks who accessed, changed, shared, deleted, or administered an item.
5. Inspect validation output when the concern is row count, freshness, duplicate keys, or exception handling.
6. Document readiness criteria and compare current evidence with those criteria before production handoff.

Use Fabric portal run history, Fabric monitoring views, Microsoft Purview audit search, query result checks, validation tables, and documented operational runbooks as evidence.

Technical Chain

A scheduled analytics solution runs under service control and emits activity state. Engines and workloads emit performance signals as they consume capacity and process data. Governance systems emit audit events for user and administrative actions. Validation logic emits data-quality evidence. Monitoring connects these signals to operational decisions. If the wrong signal is used, the team may know that something is slow but not why, or know that an item changed but not who changed it.

Operational Skills Matrix

| Task | Precise Command or Path | Verification Standard |

| ------------------------------ | -- | -- |
| Validate scheduled process health | Fabric portal > Pipeline/Dataflow/Notebook > Run or refresh history | Latest scheduled run succeeded within expected duration |

| Validate performance baseline | Fabric monitoring or workload metrics view | Current duration or utilization is compared against baseline |
| Validate audit investigation | Microsoft Purview audit search filtered to Fabric item and date | Event shows actor, action, item, and timestamp |

| Validate data-quality readiness | Query validation table or quality-check output | Freshness, counts, and exception thresholds meet criteria |

Practice Questions

1. A scheduled pipeline failed overnight. The support team needs to identify the exact failed activity and dependency before retrying. What should be inspected first?
   A. Pipeline run history and failed activity output
   B. Workspace domain assignment
   C. Item endorsement status
   D. A new capacity purchase request

2. Queries over a lakehouse Delta table have become slow after frequent small incremental writes. The evidence shows many small files. What is the most aligned optimization target?
   A. Apply row-level security
   B. Change the workspace display name
   C. Optimize the lakehouse table layout and file organization
   D. Add a sensitivity label

3. A warehouse query scans more data than necessary because it selects all columns and filters late. Which optimization should be considered first?
   A. Improve predicates and projections in the query
   B. Add a workspace role for all consumers
   C. Create a OneLake shortcut to a different folder
   D. Review pull requests in Git

4. A Spark notebook has one stage that runs far longer than the others, and execution evidence suggests skewed partitions. What should the engineer investigate?
   A. Sensitivity label inheritance
   B. Deployment pipeline history
   C. Partitioning, shuffle behavior, and skew handling in the Spark job
   D. Microsoft Purview audit search

5. A Dataflow Gen2 refresh fails after a connector credential expires. Which remediation is most precise?
   A. Update or reauthorize the source credential and rerun the refresh
   B. Increase warehouse query parallelism
   C. Create a new workspace domain
   D. Certify the dataflow item

6. An Eventstream shows growing delay between event arrival and downstream availability. What evidence should the engineer inspect?
   A. Workspace Git branch name
   B. Streaming backlog, ingestion rate, throughput, and capacity-related metrics
   C. Row-level security predicate text
   D. Manual export history

7. A production handoff requires proof that scheduled loads are healthy, data is fresh, and access changes can be investigated. What should the readiness evidence include?
   A. Only the names of workspace admins
   B. Run history, data-quality/freshness checks, metrics, and audit evidence
   C. Only the color theme of reports
   D. Only the number of notebooks in the workspace

8. A pipeline takes much longer than expected, and run history shows several independent copy activities running sequentially. What should be optimized first?
   A. Activity dependency design and supported parallelism
   B. Sensitivity label description text
   C. Workspace domain name
   D. Audit log search filters

9. A user reports that a OneLake shortcut no longer opens the target folder. What should the data engineer validate first?
   A. The shortcut target path, credential state, and permissions
   B. The deployment pipeline stage order
   C. The warehouse query text
   D. The endorsement status of the workspace

10. A manager asks whether current query duration is abnormal compared with the usual workload. Which monitoring approach is best?
    A. Apply a new sensitivity label
    B. Create a full reload pipeline immediately
    C. Change all users to Workspace Admin
    D. Compare current performance metrics or query duration against an established baseline

---

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 Implement and manage an analytics solution, Ingest and transform data, Monitor and optimize an analytics solution.

- 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: Fabric Data 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

Implement and manage an analytics solution

Q1. Correct answer: A
Explanation: The repeated behavior affects multiple items in one workspace, so the shared Spark workspace setting owns the runtime default. Row-level security controls query visibility, deployment pipelines move items between stages, and sensitivity labels classify or protect content but do not set Spark execution defaults.

Q2. Correct answer: B
Explanation: Git integration supports reviewable source changes, while deployment pipelines support controlled stage promotion. Audit logs and labels support governance, masking and object security protect data access, and shortcuts or mirroring solve data access and ingestion patterns rather than release lifecycle.

Q3. Correct answer: C
Explanation: Row-level security shapes query results based on a predicate and identity, which matches the requirement to limit visible rows. Workspace Admin is broader than needed, endorsement indicates trust but does not restrict access, and capacity sizing affects performance rather than data visibility.

Q4. Correct answer: D
Explanation: A scheduled trigger starts the process, and parameters with dynamic expressions pass runtime values into the notebook. Labels and endorsement handle governance signals, deployment comparison handles lifecycle promotion, and domain assignment organizes content but does not orchestrate runtime values.

Q5. Correct answer: D
Explanation: Domain assignment is a workspace-level governance setting, so the workspace domain configuration is the correct first inspection point. Notebook logic, retry settings, and query predicates are downstream item behaviors and do not own domain organization.

Q6. Correct answer: B
Explanation: Deployment pipeline comparison shows changed, missing, or different items across stages before deployment. Refresh history validates dataflow execution, table optimization improves storage/query performance, and dynamic masking protects values but does not compare release stages.

Q7. Correct answer: B
Explanation: The protected boundary is a column, so a column-level control is the closest match. Workspace Contributor grants broad authoring access, Eventstreams handle streaming ingestion, and Spark memory tuning affects execution performance rather than column visibility.

Q8. Correct answer: D
Explanation: Pipelines own activity ordering and runtime parameter flow, so dependency paths plus parameters match the orchestration requirement. Workspace roles, shortcuts, and labels may be useful in other scenarios, but they do not sequence activities or pass runtime values.

Q9. Correct answer: C
Explanation: Database projects support source-managed schema development and validation for database or warehouse-style objects. Dataflow staging affects preparation behavior, domains organize governance, and capacity metrics observe resource usage rather than manage schema lifecycle.

Q10. Correct answer: D
Explanation: Audit logs are the evidence source for actor, operation, item, and timestamp. Spark settings configure execution defaults, file statistics support table optimization, and retry counts help diagnose orchestration behavior but do not answer who performed an operation.

Ingest and transform data

Q1. Correct answer: D
Explanation: A watermark-based incremental pattern uses the modified timestamp to identify changed records after the prior successful load. Full reloads may be simpler but wasteful, manual copying is not a repeatable engineering pattern, and random sampling does not preserve completeness.

Q2. Correct answer: B
Explanation: Slowly changing customer attributes require a dimensional loading design that preserves current and historical state. Labels classify data, domains organize content, and capacity changes cannot replace the required transformation logic.

Q3. Correct answer: C
Explanation: A OneLake shortcut lets Fabric reference external or existing storage paths through OneLake semantics when supported. Masking protects data values, deployment stages promote items, and endorsement marks trust but does not create a storage reference.

Q4. Correct answer: B
Explanation: The failure boundary is the connector step, so source connection and credential evidence should be inspected first. Query plans, Spark skew, and deployment comparisons belong to different execution boundaries and would be lower-priority distractions.

Q5. Correct answer: D
Explanation: T-SQL is the natural transformation language for set-based warehouse joins and aggregations. Eventstreams handle streaming ingestion, sensitivity labels classify or protect assets, and domains organize governance but do not perform warehouse transformations.

Q6. Correct answer: C
Explanation: The symptom is schema drift between source and target, so the engineer must validate the changed column and adjust transformation or target schema deliberately. Capacity, endorsement, and deployment stages do not resolve the data contract mismatch.

Q7. Correct answer: D
Explanation: Continuous event capture requires a streaming pattern, commonly involving Eventstreams or Eventhouse processing depending on the Fabric design. Full reloads and manual exports are batch/manual approaches, while row-level security controls access rather than ingestion.

Q8. Correct answer: B
Explanation: Mirroring is selected when supported source data should be made available in Fabric with less custom movement logic. It does not replace security, certify items, or remove the need to validate schema, permissions, and downstream behavior.

Q9. Correct answer: A
Explanation: Duplicate output from a merge is usually owned by key selection, source deduplication, and

merge condition logic. Endorsement, audit retention, and pipeline naming do not control whether records match uniquely.

Q10. Correct answer: C
Explanation: For KQL event analysis, time filtering and selecting only required columns reduce scan scope and align the query with the scenario. Labels, database projects, and promotion workflows solve governance or lifecycle concerns rather than query selectivity.

Monitor and optimize an analytics solution

Q1. Correct answer: A
Explanation: Run history and failed activity output identify the failing boundary, dependency, and error detail. Domain assignment and endorsement are governance signals, while capacity purchase is premature before identifying the actual failure owner.

Q2. Correct answer: C
Explanation: Many small files create metadata and scan overhead, so table layout and maintenance are the correct optimization target. Row-level security and labels govern access or classification, and changing the workspace name has no performance effect.

Q3. Correct answer: A
Explanation: Query shape is the bottleneck, so predicates and projections should be improved to reduce unnecessary scans. Workspace roles, shortcuts, and Git review are useful in other contexts but do not directly reduce the inefficient query workload.

Q4. Correct answer: C
Explanation: A dominant Spark stage with suspected skew points to partitioning and shuffle behavior. Labels, deployment history, and audit search do not explain Spark execution imbalance.

Q5. Correct answer: A
Explanation: The failure owner is the source credential used by the Dataflow Gen2 refresh. Query parallelism, domains, and certification do not restore the expired connection dependency.

Q6. Correct answer: B
Explanation: Streaming delay is measured through backlog, ingestion, throughput, and capacity or workload metrics. Git branch names, RLS predicates, and manual exports do not show whether event processing is keeping up.

Q7. Correct answer: B
Explanation: Operational readiness combines execution health, quality/freshness validation, performance evidence, and auditability. Admin names, report themes, and item counts do not prove that the analytics solution is healthy or investigable.

Q8. Correct answer: A
Explanation: If independent activities run sequentially, the optimization target is orchestration design and

supported parallelism. Label text, domain names, and audit filters do not reduce avoidable pipeline wait time.

Q9. Correct answer: A
Explanation: Shortcut failures are commonly owned by target path resolution, credentials, and access permissions. Deployment stages, warehouse query text, and endorsement status are adjacent concerns but do not validate shortcut access.

Q10. Correct answer: D
Explanation: Abnormal performance requires a baseline comparison using metrics or query duration evidence. Labels classify assets, broad admin access increases risk, and a full reload pipeline is a design change that does not answer whether current duration is abnormal.