KillTest

高品質　より良いサービス



練習問題

https://www.killtest.jp

1年間無料アップデート

Exam : **DP-750**

Title : Implementing Data Engineering Solutions Using Azure Databricks

Version : DEMO

1. Topic 1, Contoso Case Study

Overview
Contoso has a single Azure Databricks workspace named Workspace1 in the West US Azure region.
Workspace1 is enabled for Unity Catalog.
Workspace1 contains all-purpose clusters for both development and production workloads.

The company's Azure environment contains:
• In the West US, Central US, and East US Azure regions, Azure event hubs that stream telemetry data and an Azure Data Lake Storage Gen2 account in each region for each hub
• A single Azure SQL database in the West US region that hosts enterprise resource planning (ERP) data
• An Azure Database for PostgreSQL server in the West US region that stores operational maintenance data

Company information
Contoso, Inc. is a renewable energy provider that operates solar and wind farms across North America.

Data Environment
Contoso ingests the following operational and business data:
• Telemetry data: More than 40,000 loT sensors across 28 sites emit JSON telemetry events every few seconds. Each site sends the events to the nearest event hub, which writes the data into the corresponding Data Lake Storage Gen2 account. These files frequently experience schema drift.
• Maintenance logs: Maintenance systems generate historical repair logs, daily incremental updates, technician notes, and unstructured attachments that are stored in the Data Lake Storage Gen2 accounts.
• Operational maintenance data: Structured operational maintenance data is stored on the Azure Database for PostgreSQL server.
• External weather data: Hourly weather forecasts are retrieved from a REST API and written to the Data Lake Storage Gen2 accounts.
• ERP data: Daily CSV extracts of 50 to 100 GB contain equipment metadata, work orders, and purchase order information.

Problem Statements
The company's existing analytics environment has several issues:

Ingestion
• Telemetry pipelines fall behind during peak loads.
• Telemetry ingestion fails when schema drift occurs.
• Streaming pipelines reprocess events after a pipeline restarts.
• Production and development workloads run on the same all-purpose clusters.
• Production and development workloads do NOT support autoscaling or workload isolation.

Governance
• The ERP data is duplicated across systems and development teams.
• Naming conventions are inconsistent across development teams, regions, and products.

• Ownership of the loT sensors changes over time, and analysts must track the full history of the ownership.
• Occasionally, equipment manufacturers must correct data-entry mistakes in equipment names. Historical values are NOT required.

Pipeline operations
• Pipelines lack resiliency, alerting, and centralized scheduling.

Planned Changes
Contoso plans to implement the following changes:
• Implement scalable data pipeline orchestration.
• Create a managed analytics catalog in Unity Catalog.
• Implement a consistent approach to creating curated datasets.
• Establish a centralized governance model across ingestion, cleansed, and curated layers.
• Grant data engineers access to the ERP tables by using minimal development effort.
• Adopt a compute strategy that isolates production workloads and supports autoscaling.
• Adopt a slowly changing dimension (SCD) approach to address current data modeling issues.

Technical Requirements
Contoso identifies the following environment and compute requirements:
• Ensure that production ingestion workloads run on compute clusters that can scale automatically during telemetry spikes.
• Provide fast and consistent performance for business intelligence (BI) workloads.
• Prevent development activity from affecting production pipelines.
• Production ingestion workloads must run as scheduled, non-interactive pipelines rather than on shared interactive development clusters.

Contoso identifies the following data ingestion and processing requirements:
• Auto-scale ingestion pipelines to handle bursty workloads.
• Handle schema drift for the maintenance and telemetry data.
• Ingest file-based telemetry data by using minimal operational effort.
• Store all the ingested data in a format that supports incremental processing.
• Support the continuous ingestion of telemetry data from the event hubs by using exactly-once semantics.
• Support the ingestion of the structured maintenance data from the Azure Database for PostgreSQL server.
• Build a new telemetry pipeline that ingests raw events from the event hubs, cleanses the data, and publishes curated tables to Unity Catalog.
• Ensure that the Apache Spark Structured Streaming pipelines reading from the event hubs write the data into a managed Delta table named telemetry.raw_events. The pipelines must support schema drift and resume processing after failures without reprocessing the data.

Contoso identifies the following data modeling and optimization requirements:
• Build curated tables that standardize business logic.

• Overwrite equipment metadata attributes, such as name, manufacturer, model, and commissioning date, when the attributes change. Historical values are NOT required.

Contoso identifies the following pipeline deployment and operation requirements:
• Orchestrate multi-step ingestion and transformation workflows.
• Define a clear execution order and dependencies.
• Automatically retry failed steps and notify operators.
• Schedule ingestion and transformation workloads consistently.

Governance Requirements
Contoso identifies the following governance requirements:
• Centralize the metadata catalog.
• Provide isolated development areas that follow standard naming conventions.
• Establish a consistent structure for organizing raw, cleansed, and curated data.
• Provide a read-only mechanism to reference the ERP data through a foreign catalog.

Business Requirements
Contoso identifies the following business requirements:
• Improve ingestion reliability and reduce operational effort.
• Standardize data definitions across development teams.

You need to develop the task logic for a new job in Lakeflow Jobs that processes telemetry data.
Each task must contain only the appropriate logic for its step in the pipeline. The solution must support the planned changes and meet the data ingestion and processing requirements.
What should you do?
A. Use a single Databricks notebook task that performs ingestion, cleansing, and curation in one script.
B. Create three tasks that each contains the identical logic and use task retries.
C. Use a single SQL task that performs ingestion, cleansing, and curation by running merge commands.
D. Create separate tasks for ingestion, cleansing, and curation.
Answer: D
Explanation:
The correct answer is D. Breaking the pipeline into separate tasks for ingestion, cleansing, and curation is the foundation of well-designed Lakeflow Jobs pipelines. Each task should own one responsibility — when a task does too much, debugging a failure becomes a hunt through unrelated code, and retry logic becomes expensive because you re-execute work that already succeeded.
Contoso's planned changes explicitly call for 'a clear execution order and dependencies' and 'orchestrate multi-step ingestion and transformation workflows.' Separate tasks map directly to those goals: Lakeflow Jobs tracks each task's status independently, so if cleansing fails, ingestion doesn't re-run.
Option A bundles everything into one notebook, which means a curation bug forces a full re-ingestion.
Option B copies logic three times — any future change must be applied in triplicate, which is a maintenance hazard.
Option C forces everything through SQL MERGE, which is the wrong tool for raw-event ingestion and doesn't address cleansing or schema drift.

2.HOTSPOT

You need to complete the PySpark code for the Spark Structured Streaming pipelines. The solution must meet the data ingestion and processing requirements.

How should you complete the code segment? To answer, select the appropriate options in the answer area. NOTE: Each correct selection is worth one point.

Answer Area

```
telemetry_path = "abfss://telemetry@contosodata.dfs.core.windows.net/"
checkpoint_path = ""
raw_events = "analytics.telemetry.raw_events"

df = (spark.readStream
          .format("json")
          .load(telemetry_path))


df.writeStream
      .format [ ]
                ("CSV")
                ("delta")
                ("json")
                ("parquet")
      .option [ ]
                ("checkpointLocation", checkpoint_path)
                ("failOnDataLoss", "false")
                ("path", checkpoint_path)
                ("startingOffsets", "earliest")
      .option("mergeSchema", "true")
      .trigger(availableNow=True)
      .table(raw_events)
```

Answer:

Answer Area

```
telemetry_path = "abfss://telemetry@contosodata.dfs.core.windows.net/"
checkpoint_path = ""
raw_events = "analytics.telemetry.raw_events"

df = (spark.readStream
        .format("json")
        .load(telemetry_path))


df.writeStream
    .format
```

```
("CSV")
("delta")
("json")
("parquet")
```

```
    .option
```

```
("checkpointLocation", checkpoint_path)
("failOnDataLoss", "false")
("path", checkpoint_path)
("startingOffsets", "earliest")
```

```
    .option("mergeSchema", "true")
    .trigger(availableNow=True)
    .table(raw_events)
```

Explanation:

The solution requires spark.readStream with format('cloudFiles') for Auto Loader, paired with .writeStream using mergeSchema=true and a checkpointLocation.

Auto Loader's cloudFiles source incrementally processes new JSON files without rescanning the entire directory. The mergeSchema option handles schema drift — when sensors add new fields, the target Delta table schema expands automatically instead of throwing a parse error. This directly addresses Contoso's requirement to 'support schema drift.'

The checkpointLocation is what gives the pipeline its resilience. Databricks writes the stream's committed offset and schema state to that path. If the cluster restarts, the engine reads the checkpoint and picks up exactly where it left off — no events are reprocessed, satisfying 'exactly-once semantics' and 'resume processing after failures without reprocessing the data.'

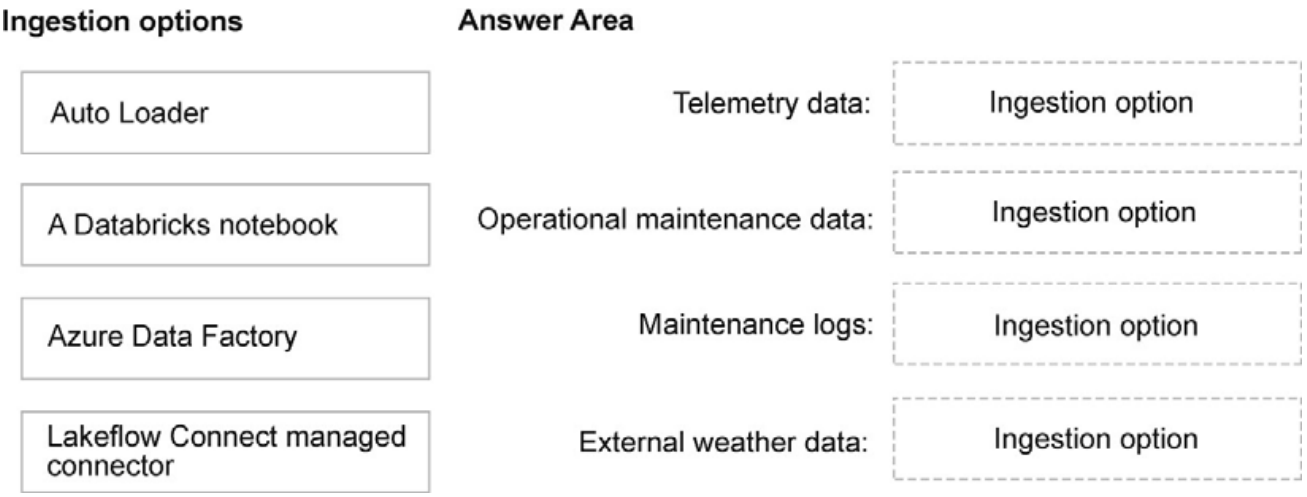
Without a checkpoint, the stream would restart from the beginning on every cluster bounce, which is precisely the problem Contoso is trying to eliminate.

Reference: https://learn.microsoft.com/en-us/azure/databricks/ingestion/auto-loader/schema

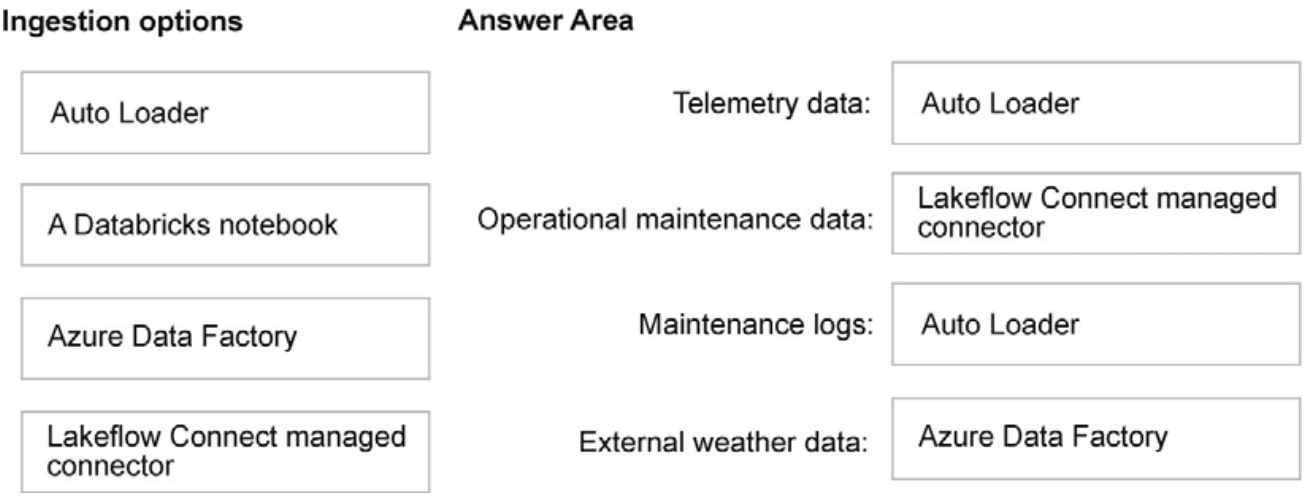
3.DRAG DROP

Which ingestion option should you recommend for each data source? To answer, drag the appropriate options to the correct data sources. Each option may be used once, more than once, or not at all. You may need to drag the split bar between panes or scroll to view content. NOTE: Each correct selection is

worth one point.



Answer:



Explanation:

The right ingestion tool depends on the source characteristics: File-based telemetry and maintenance data Auto Loader (cloudFiles). It monitors ADLS Gen2 for new arrivals, handles schema inference and evolution for the frequent schema drift, and requires minimal operational effort.

Real-time telemetry from Event Hubs Spark Structured Streaming with the azure-eventhubs-spark connector. This provides exactly-once semantics and checkpoint-based recovery, satisfying 'resume processing after failures without reprocessing.'

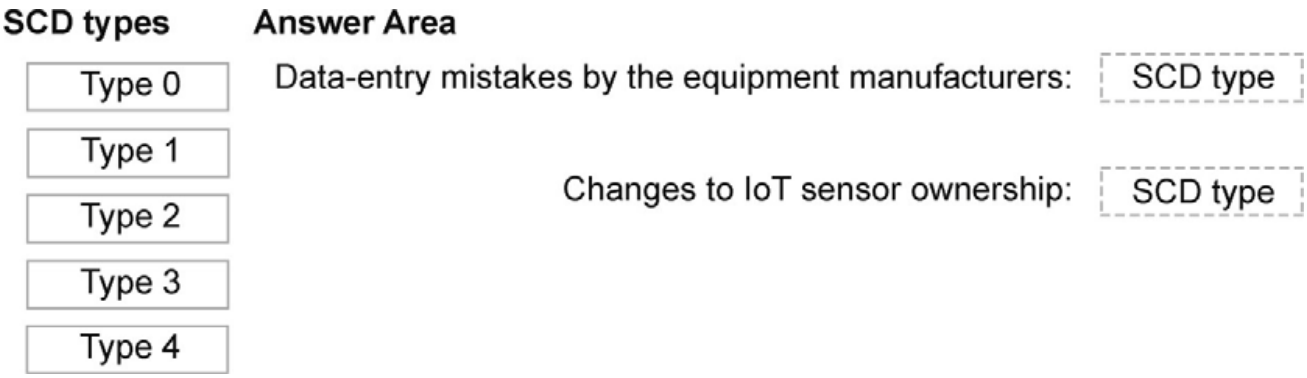
Structured maintenance data from PostgreSQL JDBC connector. Databricks supports direct JDBC reads from relational databases with pushdown predicates.

Daily CSV ERP extracts (50–100 GB) COPY INTO or Auto Loader. Both support idempotent incremental batch loading into Delta tables with minimal code.

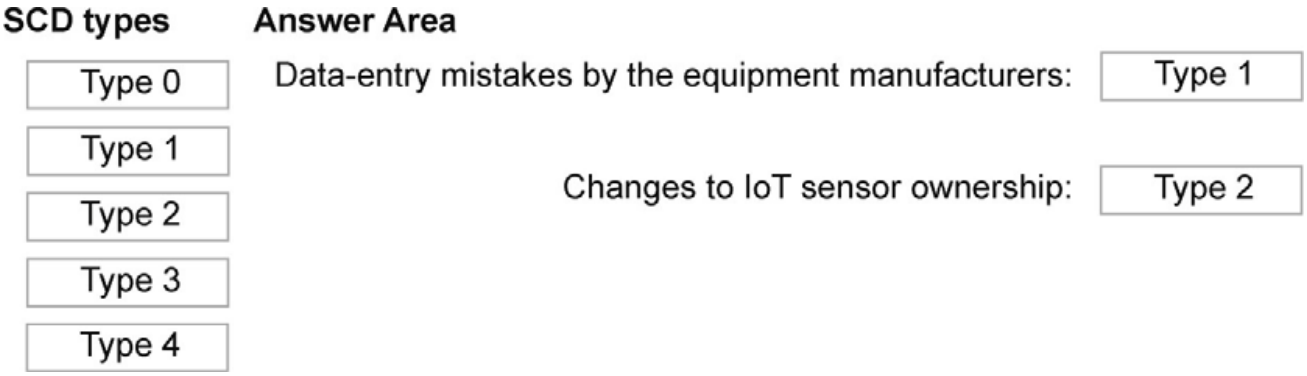
Reference: https://learn.microsoft.com/en-us/azure/databricks/ingestion/auto-loader/

4.DRAG DROP

Which SCD type should you use to support the planned data modeling changes? To answer, drag the appropriate types to the correct issues. Each type may be used once, more than once, or not at all. You may need to drag the split bar between panes or scroll to view content. NOTE: Each correct selection is worth one point.



Answer:



Explanation:

The correct mapping is SCD Type 1 for equipment metadata and SCD Type 2 for IoT sensor ownership history.

SCD Type 1 overwrites the existing record whenever an attribute changes — no history is kept. Contoso's requirement for equipment metadata (name, manufacturer, model, commissioning date) states 'historical values are NOT required,' which is the textbook definition of Type 1. A MERGE INTO with WHEN MATCHED THEN UPDATE handles this cleanly in Delta Lake.

SCD Type 2 creates a new row for each change, preserving the full history through effective-date or version columns. Contoso requires that 'analysts must track the full history of ownership' as sensors change hands over time — that full audit trail is only possible with Type 2. Type 3 (keeping just the previous value in an extra column) would lose earlier ownership records, so it doesn't satisfy the 'full history' requirement.

Reference: https://learn.microsoft.com/en-us/azure/databricks/delta/merge

5.You need to configure compute for the ingestion of telemetry data. The solution must meet the data ingestion and processing requirements.

What should you do?

A. Enable Photon acceleration for a job compute cluster.

B. Move the ingestion pipelines to shared compute.

C. Increase an all-purpose cluster to a larger fixed node type.

D. Disable autoscaling for a job compute cluster.

Answer: A

Explanation:

The correct answer is A. Photon is Azure Databricks' native vectorized query engine, written in C++, designed to accelerate data ingestion and SQL-heavy workloads significantly over the standard Spark

JVM path. Enabling it on a job compute cluster directly addresses Contoso's requirement for 'fast and consistent performance for BI workloads' and 'production ingestion workloads that can scale automatically during telemetry spikes.'
Photon integrates transparently — no code changes are needed — and pairs well with autoscaling job clusters to handle the bursty 40,000-sensor telemetry load.
Option B contradicts the isolation requirement: Contoso explicitly needs production and development separated, not merged onto shared compute.
Option C with a fixed large node gives peak capacity at all times, driving up costs even during quiet periods.
Option D disabling autoscaling is the opposite of what's needed — telemetry spikes require elastic scaling, not a locked node count.