Cloudera
CDP-3003
CDP Data Operator Certification Exam

For More Information – Visit link below:

https://www.examsboost.com/

Product Version

✓ Up to Date products, reliable and verified.
✓ Questions and Answers in PDF Format.

Latest Version: 6.0

Question: 1

You need to design a NiFi flow to process a large volume of text data (several terabytes) from multiple sources. The data contains sensitive information that needs to be anonymized before being stored in a data lake. Describe a NiFi flow that incorporates multiple processors to achieve this, including error handling, performance optimization, and security considerations. You can include code snippets for processor configuration if necessary.

A.
```python
# FlowFile Processor Configuration: ExecuteScript

# Script Language: Python

# Script:
import re

def anonymize_text(text):
  # Apply anonymization logic based on regular expressions or other methods
  # Example: Replace names with 'XXXXX'
  text = re.sub(r'[A-Z][a-z]+ [A-Z][a-z]+', 'XXXXX', text)
  return text
# Process FlowFile content:
flow_file = session.get()
# Read flow file content
text = flow_file.read()
# Anonymize text
anonymized_text = anonymize_text(text)
# Replace original content with anonymized text
flow_file = session.write(flow_file, anonymized_text)
# Transfer to next processor
session.transfer(flow_file, REL_SUCCESS)
```

B.

```
# FlowFile Processor Configuration: ReplaceText

# Search Value: 'Jane Doe'
# Replacement Value: 'XXXXX'

# Process FlowFile content:
flow_file = session.get()
# Transfer to next processor
session.transfer(flow_file, REL_SUCCESS)
```
C.
```
# FlowFile Processor Configuration: RouteOnAttribute

# Attribute: 'source'
# Routing: 'source' == 'sensitive' -> 'Anonymize'
#          'source' != 'sensitive' -> 'Success'

# Process FlowFile content:
flow_file = session.get()
# Transfer to next processor
session.transfer(flow_file, REL_SUCCESS)
```
D.
```
# FlowFile Processor Configuration: UpdateAttribute

# Attribute: 'sensitive'
# Value: 'true'

# Process FlowFile content:
flow_file = session.get()
# Transfer to next processor
session.transfer(flow_file, REL_SUCCESS)
```
E.

```
# FlowFile Processor Configuration: PutFile

# Directory: 'data_lake'
# File Name: '${filename}'

# Process FlowFile content:
flow_file = session.get()
# Transfer to next processor
session.transfer(flow_file, REL_SUCCESS)
```

Answer: A,C,D,E

Explanation:
Option A uses ExecuteScript for custom anonymization logic, offering flexibility and control. Option C routes flow files based on sensitivity, allowing for targeted anonymization. Option D tags sensitive data for tracking and potentially applying further processing. Option E stores the anonymized data in the data lake. This combination effectively handles large data volumes, incorporates error handling through the ExecuteScript processor, optimizes performance by splitting the flow, and addresses security concerns by selectively anonymizing sensitive data before storage.

Question: 2

You are processing a stream of customer data in NiFi, and you need to identify and filter out any duplicate records based on a unique customer identifier. How would you configure a NiFi flow to achieve this, ensuring minimal data loss and maintaining processing efficiency?

A. Use a single RouteOnAttribute processor to route flow files based on the customer identifier, marking duplicates for rejection. This approach allows for efficient identification of duplicates but may lead to data loss if a duplicate is encountered before the initial record is processed.
B. Employ a combination of UpdateAttribute and RouteOnAttribute processors to add a timestamp to each record, then route based on the customer identifier and timestamp, ensuring that the first occurrence of a customer record is processed while subsequent duplicates are rejected. This approach ensures minimal data loss but may impact processing efficiency.
C. Utilize a combination of UpdateAttribute and RouteOnAttribute processors to add a unique identifier to each record, then route based on this identifier, ensuring that only one instance of each record is processed. This approach ensures data integrity but may lead to unnecessary data loss if a duplicate is encountered before the initial record is processed.
D. Leverage a combination of ExecuteSQL and UpdateAttribute processors to store customer identifiers in a database table, checking for duplicates before processing each record. This approach ensures data integrity and minimal data loss but may impact processing efficiency due to database interactions.
E. Use a single UpdateAttribute processor to add a unique identifier to each record, then use a RouteOnAttribute processor to route based on this identifier, ensuring that only one instance of each

record is processed. This approach ensures data integrity but may lead to unnecessary data loss if a duplicate is encountered before the initial record is processed.

<div style="text-align: right">

Answer: D

</div>

Explanation:
Option D is the most suitable approach for this scenario. Using ExecuteSQL to store customer identifiers in a database table allows for efficient duplicate detection. By checking for duplicates before processing each record, you can ensure minimal data loss and maintain processing efficiency. Option A may lead to data loss if a duplicate arrives before the initial record is processed. Options B and C rely on timestamps or unique identifiers, which might not be robust if duplicates arrive with similar timestamps or are generated at the same time. Option E offers a similar approach to Option C, also susceptible to potential data loss due to duplicate arrival before the initial record.

Question: 3

You are building a NiFi data flow to process log files from a variety of sources. These logs contain different formats and require specific processing based on their format. Which combination of processors would be most effective for this scenario, allowing you to route data based on format and apply appropriate transformations?

A. RouteOnAttribute, UpdateAttribute, ExecuteScript
B. SplitText, RouteOnAttribute, ExecuteStreamCommand
C. GetFile, RouteOnContent, EvaluateXPath
D. GenerateFlowFile, RouteOnAttribute, UpdateAttribute
E. GetFile, RouteOnAttribute, ExecuteScript

<div style="text-align: right">

Answer: E

</div>

Explanation:
The most effective combination for this scenario is GetFile, RouteOnAttribute, ExecuteScript. * GetFile: Reads log files from various sources. RouteOnAttribute: Routes the flow files based on the format (identified through an attribute like 'logFormat'). ExecuteScript: Executes custom scripts for specific formats, applying appropriate transformations. This approach allows for flexible routing and processing based on the content of the log files, making it ideal for handling diverse formats. Other Options: A: While useful, UpdateAttribute is not a primary routing mechanism. * B: SplitText may not be needed if logs are already in separate files. ExecuteStreamCommand is not suitable for format-specific transformations. C: RouteOnContent is inefficient for large log files. EvaluateXPath is for XML processing, not general log formats. * D: GenerateFlowFile is not needed here. UpdateAttribute is not the primary routing mechanism.

Question: 4

You are working with a data flow that processes large CSV files with millions of rows. The flow needs to process the data in batches to manage memory usage and improve performance. Which processor

would be most suitable for creating these batches, while ensuring data order is preserved within each batch?

A. SplitText
B. SplitContent
C. SplitRecord
D. PartitionRecord
E. MergeContent

Answer: C

Explanation:
The most suitable processor for batching large CSV files while preserving data order is SplitRecord. SplitRecord: Splits the content of a flow file into multiple flow files based on records (lines in this case), allowing you to process them in smaller batches. It also maintains the order of records within each batch. Other Options: A: SplitText is not designed for record-based splitting, it splits based on delimiters. * B: SplitContent splits based on content, not records, leading to inconsistent batch sizes and potential order issues. * D: PartitionRecord is for splitting based on attributes, not content. E: MergeContent combines flow files, the opposite of what's needed here.

Question: 5

You're processing real-time data from a streaming source using NiFi. The data needs to be aggregated over a specific time window (e.g., 5 minutes) before being written to a database. How would you implement this aggregation and time-based grouping using NiFi processors?

A. Use the 'MergeContent' processor with a custom aggregation script.
B. Utilize the 'Groupcontent' processor with 'TimeWindow' set to 5 minutes.
C. Employ the 'UpdateAttribute' processor to add a timestamp attribute and then use 'RouteOnAttribute' to group by time intervals.
D. Combine 'ExecuteScript' with a custom script for aggregation and 'RouteOnAttribute' for time-based grouping.
E. Implement a custom NiFi processor using the NiFi API to achieve the required aggregation and time window functionality.

Answer: B

Explanation:
The most efficient and effective solution is to use the GroupContent processor with TimeWindow set to 5 minutes. * Groupcontent: This processor groups flow files based on various criteria, including time windows. It allows for data aggregation within each time window, providing the desired functionality. Other Options: * A: MergeContent is for merging flow files, not for time-based aggregation. * C: While UpdateAttribute adds timestamps, it doesn't automatically group data into time windows. * D: This approach is more complex and might not be as efficient as GroupContent. * E: While possible, developing a custom processor is an overkill for this specific need.

Question: 6

Your NiFi data flow is designed to process sensitive dat
a. You need to ensure that the data is encrypted during transmission between processors. Which processor would be most suitable for this purpose?
A. EncryptContent
B. DecryptContent
C. EncryptAttribute
D. DecryptAttribute
E. UpdateAttribute

Answer: A

Explanation:
The EncryptContent processor is specifically designed to encrypt the content of flow files. It uses a variety of encryption algorithms and key management mechanisms to secure the data during transmission. Other Options: B: DecryptContent is for decrypting previously encrypted content. * C: EncryptAttribute encrypts individual attributes, not the entire content. * D: DecryptAttribute decrypts attributes. * E: UpdateAttribute is for modifying attributes, not for encryption.

Question: 7

You are processing data from a web service that provides data in JSON format. You need to extract specific fields from the JSON and use them to update attributes of the flow file. Which processor would be most effective for this task?

A. EvaluateXPath
B. ExecuteScript
C. SplitJson
D. UpdateAttribute
E. RouteOnAttribute

Answer: B

Explanation:
The ExecuteScript processor is the most versatile and effective option for extracting specific fields from JSON data. You can write a custom script (e.g., using Groovy) to parse the JSON and extract the desired fields. These fields can then be used to update attributes using the Update Attribute processor. Other Options: A: EvaluateXPath is for XML processing, not JSON. * C: SplitJson splits JSON into separate flow files, not for field extraction. * D: UpdateAttribute modifies attributes, but it doesn't extract data from JSON. * E: RouteOnAttribute routes flow files based on attributes, not for data extraction.

Question: 8

You need to design a NiFi data flow to ingest data from a Kafka topic, transform the data, and write it to a Hadoop HDFS directory. The data flow should be resilient and handle potential failures during processing. Which processor would be most suitable for ensuring fault tolerance and data delivery guarantee in this scenario?

A. GetFile
B. PutFile
C. ListenHTTP
D. ListenKafka
E. ExecuteStreamCommand

Answer: D

Explanation:
The ListenKafka processor is specifically designed for consuming data from Kafka topics. It provides built-in fault tolerance mechanisms, including: Consumer Groups: Ensures that only one instance of NiFi processes a specific partition of a Kafka topic, preventing duplicate processing. * Offset Management: Tracks the last processed message and automatically resumes from the last committed offset in case of failures. * Retry Mechanisms: Handles transient errors and retries message processing until success. Other Options: * A: GetFile reads files from a file system. * B: PutFile writes files to a file system. C: ListenHTTP listens for HTTP requests. * E: ExecuteStreamCommand is for executing external commands, not for handling Kafka data.

Question: 9

You are building a data flow to process time-sensitive events from a high-volume streaming source. The flow needs to handle bursts of
data without dropping events and ensure timely processing. Which NiFi processor would be most effective for managing these high-
volume, time-sensitive events while preventing data loss?

A. RouteOnAttribute
B. UpdateAttribute
C. SplitContent
D. MergeContent
E. Queue

Answer: E

Explanation:
The Queue processor is crucial for managing high-volume, time-sensitive events. It acts as a buffer, allowing you to control the flow of data and prevent data loss during bursts. By providing a temporary holding area, the Queue ensures that events are processed as quickly as possible, even when the downstream processing capacity is limited. Other Options: * A: RouteOnAttribute routes based on attributes, not for managing high volumes. B: UpdateAttribute updates attributes, not for volume

management. * C: SplitContent splits content, not for managing high volumes. * D: MergeContent merges content, not for managing high volumes.

Question: 10

You have a flow in NiFi that processes a large stream of data containing customer information. Each customer record is represented as a JSON object. You need to enrich the data with additional information from an external API that provides customer demographics. The API requires a unique customer ID as input. Which NiFi processors are most suitable for this scenario, and in what order should they be used?

A. InvokeHTTP, UpdateAttribute
B. GetFile, SplitJson, ExecuteScript
C. RouteOnAttribute, MergeContent, UpdateAttribute
D. ExtractText, EvaluateJsonPath, PutDatabaseRecord
E. GenerateFlowFile, ReplaceText, UpdateAttribute

Answer: A

Explanation:
The correct answer is A. In this scenario, you'll need to use the 'InvokeHTTP' processor to make requests to the external API, passing the customer ID as a parameter. The UpdateAttribute& processor is used to add the retrieved demographic data as new attributes to the flow file. Option B focuses on splitting and manipulating JSON, not API interaction. Option C involves routing based on attributes, which is not the primary requirement. Option D deals with extracting and storing data in a database. Option E generates flow files, which is not necessary. The ' InvokeHTTP and 'UpdateAttribute' processors provide a streamlined approach to enrich the flow files with data from an external API.

Thank You for Trying Our Product

For More Information – Visit link below:

https://www.examsboost.com/

15 USD Discount Coupon Code:

G74JA8UF

FEATURES

- ✓ **90 Days Free Updates**

- ✓ **Money Back Pass Guarantee**

- ✓ **Instant Download or Email Attachment**

- ✓ **24/7 Live Chat Support**

- ✓ **PDF file could be used at any Platform**

- ✓ **50,000 Happy Customer**