DUMPSARENA

Confluent Certified Administrator for Apache Kafka Certification Examination

Confluent CCAAK

Version Demo

Total Demo Questions: 7

Total Premium Questions: 75

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Apache Kafka Fundamentals	36
Topic 2, Apache Kafka Cluster Configuration	15
Topic 3, Apache Kafka Security	12
Topic 4, Apache Kafka Monitoring and Management	5
Topic 5, Kafka Connect	7
Total	75

QUESTION NO: 1

Which use cases would benefit most from continuous event stream processing? (Choose three.)

A. Fraud detection

B. Context-aware product recommendations for e-commerce

C. End-of-day financial settlement processing

D. Log monitoring/application fault detection

E. Historical dashboards

ANSWER: A B D

Explanation:

Continuous event stream processing is most valuable where incoming events must be evaluated and acted on with low latency, rather than waiting for a scheduled batch run. **Fraud detection** uses continuous processing to evaluate payment, login, or account activity as it occurs, correlate it with recent behavior, and identify suspicious patterns quickly enough to prevent or limit loss. **Context-aware product recommendations for e-commerce** benefit because clicks, searches, cart changes, and purchases can immediately update customer context and trigger relevant recommendations during an active session. **Log monitoring/application fault detection** is also a core streaming use case: operational events can be continuously filtered, aggregated, and correlated to detect error-rate spikes, failed requests, or anomalous system behavior and produce timely alerts. Kafka Streams supports stateful, real-time processing directly against Kafka topics, including joins, aggregations, and time-windowed computations. Confluent describes stream processing as transforming, enriching, and analyzing event streams in motion, making these low-latency detection and response workloads especially appropriate. See Confluent Platform documentation for Kafka Streams and Confluent documentation on ksqlDB concepts.

QUESTION NO: 2

Your Kafka cluster has four brokers. The topic t1 on the cluster has two partitions, and it has a replication factor of three. You create a Consumer Group with four consumers, which subscribes to t1.

In the scenario above, how many Controllers are in the Kafka cluster?

A. One

B. Two

C. Three

D. Four

ANSWER: A

Explanation:

One is correct. A Kafka cluster has exactly one active controller at a time. The controller role is held by one broker in ZooKeeper-based Kafka clusters; in KRaft-mode clusters, it is held by one active controller node (which can be a broker in a combined broker/controller deployment). The active controller coordinates cluster-wide metadata and control-plane operations, including managing broker membership and initiating partition leader changes when needed. Kafka provides redundancy through controller election: if the active controller becomes unavailable, another eligible controller is elected, but this does not result in multiple active controllers simultaneously.

The number of brokers, topic partitions, replication factor, and consumers in a consumer group do not alter this fundamental one-active-controller design. In this scenario, the four brokers host replicas and may be eligible for the controller role depending on the deployment configuration, but only one is the currently active controller. The two partitions determine the maximum number of consumers that can actively receive assigned partitions from this topic, while replication factor three determines how many broker replicas each partition has; neither setting changes the controller count.

See Confluent's KRaft overview and Apache Kafka's KRaft controller documentation.

QUESTION NO: 3

What is the atomic unit of data in Kafka?

A. Partition

B. Message

C. Topic

D. Offset

ANSWER: B

Explanation:

Message is correct. In Kafka terminology, the atomic unit written to and read from a topic partition is commonly called a message; current Kafka documentation also uses the more precise term *record*. A message/record contains the data payload (value) and can include a key, timestamp, and headers. Producers publish these individual units to a topic, Kafka appends them to one of that topic's partitions, and consumers retrieve them as records. Each message is assigned a position within its partition, represented by an offset, which enables consumers to track their progress reliably. This distinction is important operationally: a topic is the named stream, partitions are the ordered logs that provide Kafka's scalability and parallelism, and offsets identify the position of individual messages within those logs. The message itself is therefore the indivisible data item that Kafka stores and transfers. Kafka's protocol documentation describes records as the units in record batches, while the introductory documentation explains that producers write events (records) to Kafka topics. See the Apache Kafka introduction and core concepts and the Apache Kafka record/message format documentation.

QUESTION NO: 4

What is the relationship between topics and partitions? (Choose two.)

A. A topic always has one partition.

B. A topic may have more than one partition.

C. A partition is always linked to a single topic.

D. A partition may have more than one topic.

E. There is no relationship between topics and partitions.

ANSWER: B C

Explanation:

A topic may have more than one partition. In Kafka, a topic is a logical category or stream of records that is divided into one or more partitions. Partitions provide the unit of storage, ordering, and parallelism for the topic. Configuring multiple partitions lets Kafka distribute a topic's data across brokers and enables multiple consumers in the same consumer group to process records from that topic in parallel, subject to the number of available partitions. A topic can also be created with a single partition when its throughput and parallel-consumption needs are limited.

A partition is always linked to a single topic. More specifically, each partition is an ordered, immutable sequence of records within one particular topic, identified by its topic name and partition number. Records are appended to that partition and receive offsets that are meaningful in the context of that partition. This topic-to-partition structure is fundamental to Kafka's data model and allows topics to scale while retaining ordering within an individual partition. See the Apache Kafka documentation on topics and partitions and Confluent's Kafka introduction.

QUESTION NO: 5

Which out-of-the-box Kafka Authorizer implementation uses ZooKeeper?

A. RBAC

B. ACLs

C. Ranger

D. LDAP

E. SimpleAclAuthorizer

ANSWER: E

Explanation:

`SimpleAclAuthorizer` is the out-of-the-box Apache Kafka authorizer implementation historically used with ZooKeeper-based Kafka clusters. It persists Kafka ACL data in ZooKeeper and loads that ACL information to make authorization decisions for client requests. This is the specific implementation name sought by the question; ACLs are the authorization rules managed by the authorizer, rather than the authorizer implementation itself.

Kafka's security documentation identifies `SimpleAclAuthorizer` as the ZooKeeper-backed implementation and documents its configuration through the `authorizer.class.name` broker property. For a ZooKeeper-based deployment, configuring this class enables Kafka's built-in ACL authorization model, allowing permissions to be granted for resources such as topics, consumer groups, transactional IDs, and cluster operations. See the Apache Kafka Authorization and ACL documentation and the Kafka API documentation for the ACL authorizer. Newer KRaft-based Kafka deployments use a different metadata architecture, but the ZooKeeper-specific implementation referred to here is `SimpleAclAuthorizer`.

QUESTION NO: 6

The Consumer property â€˜auto offset resetâ€™ determines what to do if there is no valid offset for a Consumer Group.

Which scenario is an example of a valid offset and therefore the â€˜auto.offset.resetâ€™ does NOT apply?

A. The Consumer offset is greater than the last offset in the partition (log end offset).

B. The Consumer offset is less than the smallest offset in the partition (log start offset).

C. The Consumer Group started for the first time.

D. When an offset points to a message that has been removed by compaction but is still within the current partition.offset range.

ANSWER: D

Explanation:

When an offset points to a message that has been removed by compaction but is still within the current partition.offset range is correct because Kafka compaction does not renumber offsets or remove their positions from the partition's offset sequence. A compacted record may no longer be returned to a consumer, but the consumer group's committed position can still be within the partition's retained log range. Kafka therefore treats that committed position as valid and resumes consumption from it; it does not invoke `auto.offset.reset`. The consumer will simply advance through the log as needed until it reaches an available record.

The `auto.offset.reset` setting is a fallback policy used only when the group has no committed offset for the partition or when its committed offset is out of range. In contrast, compaction affects the presence of individual records while preserving the logical offset structure needed for consumer positions. This distinction is important when consuming compacted topics such as changelog or state topics: an absent historical record does not by itself mean that the group's stored offset is invalid. See the Apache Kafka consumer configuration definition for auto.offset.reset and Confluent's explanation of log compaction.

QUESTION NO: 7

Why does Kafka use ZooKeeper? (Choose two.)

A. To access information about the leaders and partitions

B. To scale the number of brokers in the cluster

C. To prevent replication between clusters

D. For controller election

ANSWER: A D

Explanation:

In ZooKeeper-based Kafka clusters, ZooKeeper provides the distributed coordination and metadata services needed for the brokers to operate as a cluster. **To access information about the leaders and partitions** is correct because ZooKeeper stores cluster metadata used by brokers, including broker registrations and topic/partition state. Kafka's controller maintains partition leadership and replica-assignment information and propagates the resulting metadata to brokers, enabling requests to be routed to the current partition leader.

For controller election is also correct because ZooKeeper elects one broker as the active controller. The controller is responsible for cluster-wide coordination activities, such as reacting to broker changes and managing partition leader transitions. ZooKeeper's ephemeral znodes and watch mechanism allow Kafka to detect a controller or broker failure and establish a replacement controller when necessary.

This describes Kafka's legacy ZooKeeper mode. Modern Kafka clusters can instead use KRaft, Kafka's built-in metadata quorum, which removes the ZooKeeper dependency; however, ZooKeeper remains relevant for existing ZooKeeper-mode deployments. See the Apache Kafka documentation on ZooKeeper-based Kafka and Confluent's overview of KRaft mode.