Demonstration: run-kafka-blockchain.demo.sh

1. Demonstrate putting messages into a Kafka blockchain with one partition, one producer and one consumer.
2. Describe the single blockchain partition
3. Demonstrate verifying messages

1. Demonstrate putting four tamper-proof messages into a Kafka blockchain named kafka-demo-blockchain.

This demonstration takes advantage of the immutable records feature recently introduced by Java version 14. Consequently, the --enable-preview option must be specified.

reed@gold:~/KafkaBlockchain$ scripts/run-kafka-blockchain-demo.sh
Picked up _JAVA_OPTIONS: -ea -Xms1G -Xmx5G --enable-preview

These warnings are from a Maven dependent Java library and can be ignored.

WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.google.inject.internal.cglib.core.$ReflectUtils$1 (file:/usr/share/maven/lib/guice.jar) to method java.lang.ClassLoader.defineClass(java.lang.String,byte[],int,int,java.security.ProtectionDomain)
WARNING: Please consider reporting this to the maintainers of com.google.inject.internal.cglib.core.$ReflectUtils$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release

This demonstration uses ZooKeeper to store genesis hashes for named blockchains. Production KakaBlockchain applications could use an encrypted secret store.

The below console logs have been simplified with timestamps removed. The format is process thread name, [Java class name], comment text.

main [KafkaBlockchainDemo] **connecting to ZooKeeper...**
main [ZooKeeperAccess] connection to ZooKeeper at **ip6-localhost:2181**
main-EventThread [ZooKeeperAccess] watchedEvent: WatchedEvent state:SyncConnected type:None path:null
main-EventThread [ZooKeeperAccess] ZooKeeper session is connected

This step activates Kafka messaging and a consumer process thread. Each blockchain is identified by topic. This demonstration configures a single Kafka partition so that records are sequentially ordered, which is a blockchain algorithm requirement. With multiple partitions, demonstrated later, parallel producer process threads cooperate to sequentially blockchain records into multiple partitions.

The main process thread initializes the Kafka topic.
The kafkaConsumer process thread begins polling for records in its topic.

main [KafkaBlockchainDemo] **activating Kafka messaging**
main [KafkaBlockchainDemo] Kafka topics [kafka-demo-blockchain]
main [KafkaBlockchainDemo$ConsumerLoop] consuming messages for Kafka topic kafka-demo-blockchain
main [KafkaBlockchainDemo] now consuming messages from topic kafka-demo-blockchain
kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] subscribing to [kafka-demo-blockchain]
kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] consumer loop poll...

This step, performed entirely in the main process thread, creates the genesis (first) record of the blockchain, and stores the SHA-256 hash value in ZooKeeper, to be retrieved by the blockchain name, which is the Kafka topic. Tamper-evident records are created by library methods that wrap the binary payload bytes in a Java 14 immutable record.

This tamper-evident record contains as fields:

1. *The payload bytes, typically a serialized Java object.*
2. *The SHA-256 hash value of the previous record in the blockchain.*
3. *The serial number.*
4. *The SHA-256 hash value of this record: payload, previous block hash, serial number.*

The tamper-evident record displays as [TEObject serial-number, wrapping a payload of NNN bytes]

main [KafkaBlockchainDemo] initial previousTEObjectHash: [KafkaBlockchainInfo kafka-demo-blockchain, serial 1, hash 55800d7983a2a9904d34a306a176e303c5cd2b75dd99b8a761e92140ef9d4510, timestamp Wed Jul 08 19:52:00 MDT 2020
main [KafkaBlockchainDemo] **genesis hash** for kafka-demo-blockchain=cc715cc85ed5e7dee99c2468b3e5f34afee812e6f50faa12aa17280425c5f6f1
main [ZooKeeperAccess] setDataString cc715cc85ed5e7dee99c2468b3e5f34afee812e6f50faa12aa17280425c5f6f1 at path /KafkaBlockchain/kafka-demo-blockchain
main [ZooKeeperAccess] creating path /KafkaBlockchain
main [ZooKeeperAccess] actualPath /KafkaBlockchain
main [ZooKeeperAccess] creating path /KafkaBlockchain/kafka-demo-blockchain
main [ZooKeeperAccess] setDataString cc715cc85ed5e7dee99c2468b3e5f34afee812e6f50faa12aa17280425c5f6f1 at path /KafkaBlockchain/kafka-demo-blockchain
main [ZooKeeperAccess] actualPath /KafkaBlockchain/kafka-demo-blockchain
main [KafkaBlockchainDemo] **sent producer record** [TEObject 1, wrapping a payload of 232 bytes]

The main process thread continues creating tamper-evident records for three more sample payloads.

main [KafkaBlockchainDemo] **sent producer record** [TEObject 2, wrapping a payload of 232 bytes]
main [KafkaBlockchainDemo] **sent producer record** [TEObject 3, wrapping a payload of 232 bytes]
main [KafkaBlockchainDemo] **sent producer record** [TEObject 4, wrapping a payload of 232 bytes]

The kafkaConsumer process thread polls the topic and receives the blockchain records in sequence. The tamper-evident record is deserialized and the payload obtained, without recomputing the hash chain - which is slightly more complicated and demonstrated in a separate script below.

The application use case situation determines whether to spend the effort verifying a particular blockchain. One alternative is to always verify it by spending the time to re-compute the SHA-256 hash from the genesis block or from the most recent checkpoint block. Another is to periodically perform the verification audit according to risk and consequences of corruption.

kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] **received consumerRecord** Wed Jul 08 19:52:00 MDT 2020
kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] deserialized teObject [TEObject 1, wrapping a payload of 232 bytes]
kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] **payload: [DemoPayload, string=abc, integer=1]**
kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] consumer loop poll...

kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] **received consumerRecord** Wed Jul 08 19:52:00 MDT 2020
kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] deserialized teObject [TEObject 2, wrapping a payload of 232 bytes]
kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] payload: [DemoPayload, string=def, integer=2]

kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] **received consumerRecord** Wed Jul 08 19:52:00 MDT 2020
kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] deserialized teObject [TEObject 3, wrapping a payload of 232 bytes]
kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] payload: [DemoPayload, string=ghi, integer=3]

kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] **received consumerRecord** Wed Jul 08 19:52:00 MDT 2020
kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] deserialized teObject [TEObject 4, wrapping a payload of 232 bytes]
kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] payload: [DemoPayload, string=jkl, integer=4]
kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] consumer loop poll...

The kafkaConsumer process thread is done reading tamper-evident records from the blockchain and shuts down.

kafkaConsumer [KafkaBlockchainDemo$ConsumerLoop] closing the consumer loop...

The main process thread shuts down this demonstration.

main [ZooKeeperAccess] closing the ZooKeeper connection ...
main-EventThread [ZooKeeperAccess] watchedEvent: WatchedEvent state:Closed type:None path:null

2. Describe the blockchain partitions for the Kafka group demo-blockchain-consumers

reed@gold:~/KafkaBlockchain$ ${HOME}/$KAFKA_VERSION/bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --group demo-blockchain-consumers --describe
Picked up _JAVA_OPTIONS: -ea -Xms1G -Xmx5G --enable-preview

Consumer group 'demo-blockchain-consumers' has no active members.

This step uses a Kafka built in utility to describe the kafka-demo-blockchain statistics for the single partition.

GROUP	TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG	CONSUMER-ID	HOST	CLIENT-ID
demo-blockchain-consumers	kafka-demo-blockchain	0	4	4	0	-	-	-

3. Demonstrate verifying messages from the Kafka blockchain named kafka-demo-blockchain

This demonstration retrieves the genesis hash from ZooKeeper, then reads all the blockchain tamper-evident records from the blockchain topic, and compares their expected SHA-256 hashes with the recomputed hashes in sequence.

reed@gold:~/KafkaBlockchain$ mvn "-Dexec.args=-classpath %classpath com.ai_blockchain.kafka_bc.samples.KafkaBlockchainDemoVerification kafka-demo-blockchain" -Dexec.executable=java -Dexec.classpathScope=runtime org.codehaus.mojo:exec-maven-plugin:1.5.0:exec
Picked up _JAVA_OPTIONS: -ea -Xms1G -Xmx5G --enable-preview

The main process thread connects to ZooKeeper and retrieves the genesis SHA-256 hash for the demo blockchain topic.

main [KafkaBlockchainDemoVerification] Verifying the specified Kafka blockchain named kafka-demo-blockchain
main [KafkaBlockchainDemoVerification] connecting to ZooKeeper...
main [KafkaBlockchainDemoVerification] Kafka topics [kafka-demo-blockchain]
main [KafkaBlockchainDemoVerification] genesis hash for path /KafkaBlockchain/kafka-demo-blockchain = cc715cc85ed5e7dee99c2468b3e5f34afee812e6f50faa12aa17280425c5f6f1

The main process thread polls for records from the demo blockchain topic.

main [KafkaBlockchainDemoVerification] now consuming messages from topic kafka-demo-blockchain
main [KafkaBlockchainDemoVerification] subscribing to [kafka-demo-blockchain]
main [KafkaBlockchainDemoVerification] consumer loop poll...

The main process thread receives the first tamper-evident record from the blockchain, and verifies that this genesis record SHA-256 hash matches the expected value that was retrieved from ZooKeeper for this blockchain topic.

main [KafkaBlockchainDemoVerification] **received consumerRecord** ConsumerRecord(topic = kafka-demo-blockchain, partition = 0, leaderEpoch = 0, offset = 0, CreateTime = 1594259520554, serialized key size = -1, serialized value size = 597, headers = RecordHeaders(headers = [], isReadOnly = false), key = null, value = [B@41fecb8b)
main [KafkaBlockchainDemoVerification] deserialized teObject [TEObject 1, wrapping a payload of 232 bytes]
main [KafkaBlockchainDemoVerification] **payload: [DemoPayload, string=abc, integer=1]**
main [KafkaBlockchainDemoVerification] **the genesis record verifies with the expected SHA-256 hash**

The main process thread receives the next three tamper-evident records from the blockchain, and verifies that each record SHA-256 hash matches the expected value that is recomputed at this point in the sequence.

main [KafkaBlockchainDemoVerification] **received consumerRecord** ConsumerRecord(topic = kafka-demo-blockchain, partition = 0, leaderEpoch = 0, offset = 1, CreateTime = 1594259520565, serialized key size = -1, serialized value size = 597, headers = RecordHeaders(headers = [], isReadOnly = false), key = null, value = [B@f736069)
main [KafkaBlockchainDemoVerification] deserialized teObject [TEObject 2, wrapping a payload of 232 bytes]
main [KafkaBlockchainDemoVerification] payload: [DemoPayload, string=def, integer=2]
main [KafkaBlockchainDemoVerification] **this record is a valid successor** in the kafka-demo-blockchain blockchain

main [KafkaBlockchainDemoVerification] **received consumerRecord** ConsumerRecord(topic = kafka-demo-blockchain, partition = 0, leaderEpoch = 0, offset = 2, CreateTime = 1594259520569, serialized key size = -1, serialized value size = 597, headers = RecordHeaders(headers = [], isReadOnly = false), key = null, value = [B@2fb3536e)
main [KafkaBlockchainDemoVerification] deserialized teObject [TEObject 3, wrapping a payload of 232 bytes]
main [KafkaBlockchainDemoVerification] payload: [DemoPayload, string=ghi, integer=3]
main [KafkaBlockchainDemoVerification] **this record is a valid successor** in the kafka-demo-blockchain blockchain

main [KafkaBlockchainDemoVerification] **received consumerRecord** ConsumerRecord(topic = kafka-demo-blockchain, partition = 0, leaderEpoch = 0, offset = 3, CreateTime = 1594259520569, serialized key size = -1, serialized value size = 597, headers = RecordHeaders(headers = [], isReadOnly = false), key = null, value = [B@3b6ddd1d)
main [KafkaBlockchainDemoVerification] deserialized teObject [TEObject 4, wrapping a payload of 232 bytes]
main [KafkaBlockchainDemoVerification] payload: [DemoPayload, string=jkl, integer=4]
main [KafkaBlockchainDemoVerification] **this record is a valid successor** in the kafka-demo-blockchain blockchain

The main process thread shuts down this demonstration.

main [KafkaBlockchainDemoVerification] ...end of consumed records
main [KafkaBlockchainDemoVerification] closing the consumer loop...

Demonstration: run-kafka-blockchain-multiple-partition.demo.sh

1. Demonstrate putting messages into a Kafka blockchain with five partitions, one producer and one consumer.
2. Describe the five blockchain partitions
3. Demonstrate verifying messages with five cooperating consumers - one for each partition.

1. Demonstrate putting messages into a Kafka blockchain named kafka-demo-multiple-partition-blockchain

This demonstration takes advantage of the immutable records feature recently introduced by Java version 14. Consequently, the --enable-preview option must be specified.

```
Picked up _JAVA_OPTIONS: -ea -Xms1G -Xmx5G --enable-preview
```

This demonstration uses ZooKeeper to store genesis hashes for named blockchains. Production KakaBlockchain applications could use an encrypted secret store.

The below console logs have been simplified with timestamps removed. The format is process thread name, [Java class name], comment text.

```
main [KafkaBlockchainMultiplePartitionDemo] connecting to ZooKeeper...
```

This step activates Kafka messaging and a consumer process thread. Each blockchain is identified by topic. This demonstration configures five Kafka partitions with one producer process thread. Should the application require, parallel producer process threads must cooperate and synchronize to assign serial numbers to blockchain records.

The main process thread initializes the Kafka topic.

```
main [KafkaBlockchainMultiplePartitionDemo] activating Kafka messaging
main [KafkaBlockchainMultiplePartitionDemo]   Kafka topics [kafka-demo-blockchain,
kafka-demo-multiple-partition-blockchain]
```

The kafkaConsumer process thread begins polling for records in its topic.

```
main [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] consuming messages for Kafka topic
kafka-demo-multiple-partition-blockchain
main [KafkaBlockchainMultiplePartitionDemo] now consuming messages from topic
kafka-demo-multiple-partition-blockchain
kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] subscribing to
[kafka-demo-multiple-partition-blockchain]
kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] consumer loop poll...
```

This step, performed entirely in the main process thread, creates the genesis (first) record of the blockchain, and stores the SHA-256 hash value in ZooKeeper, to be retrieved by the blockchain name, which is the Kafka topic. Tamper-evident records are created by library methods that wrap the binary payload bytes in a Java 14 immutable record.

This tamper-evident record contains as fields:

5. *The payload bytes, typically a serialized Java object.*
6. *The SHA-256 hash value of the previous record in the blockchain.*
7. *The serial number.*
8. *The SHA-256 hash value of this record: payload, previous block hash, serial number.*

The tamper-evident record displays as [TEObject serial-number, wrapping a payload of NNN bytes]

main [KafkaBlockchainMultiplePartitionDemo] initial previousTEObjectHash: [KafkaBlockchainInfo kafka-demo-multiple-partition-blockchain, serial 1, hash 55800d7983a2a9904d34a306a176e303c5cd2b75dd99b8a761e92140ef9d4510, timestamp Wed Jul 08 20:25:14 MDT 2020
main [KafkaBlockchainMultiplePartitionDemo] **genesis hash** for kafka-demo-multiple-partition-blockchain=3e9801a24dbe3f19e0ae9430dd9dd942cb39f9b1431c25a56c63d8a0c8b0661e
main [ZooKeeperAccess] setDataString 3e9801a24dbe3f19e0ae9430dd9dd942cb39f9b1431c25a56c63d8a0c8b0661e at path /KafkaBlockchain/kafka-demo-multiple-partition-blockchain
main [KafkaBlockchainMultiplePartitionDemo] **sent producer record** [TEObject 1, wrapping a payload of 249 bytes]

The main process thread continues creating tamper-evident records for four more sample payloads.

main [KafkaBlockchainMultiplePartitionDemo] **sent producer record** [TEObject 2, wrapping a payload of 249 bytes]
main [KafkaBlockchainMultiplePartitionDemo] **sent producer record** [TEObject 3, wrapping a payload of 249 bytes]
main [KafkaBlockchainMultiplePartitionDemo] **sent producer record** [TEObject 4, wrapping a payload of 249 bytes]
main [KafkaBlockchainMultiplePartitionDemo] **sent producer record** [TEObject 5, wrapping a payload of 249 bytes]

The kafkaConsumer process thread polls the topic and receives the blockchain records in sequence. The tamper-evident record is deserialized and the payload obtained, without recomputing the hash chain - which is slightly more complicated and demonstrated in a separate script below.

The application use case situation determines whether to spend the effort verifying a particular blockchain. One alternative is to always verify it by spending the time to re-compute the SHA-256 hash from the genesis block or from the most recent checkpoint block. Another is to periodically perform the verification audit according to risk and consequences of corruption.

kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] **received consumerRecord** Wed Jul 08 20:25:14 MDT 2020
kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] deserialized teObject [TEObject 1, wrapping a payload of 249 bytes]
kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] **payload: [DemoPayload, string=abc, integer=1]**
kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] consumer loop poll...

kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] **received consumerRecord** Wed Jul 08 20:25:14 MDT 2020
kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] deserialized teObject [TEObject 2, wrapping a payload of 249 bytes]
kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] payload: [DemoPayload, string=def, integer=2]

kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] **received consumerRecord** Wed Jul 08 20:25:14 MDT 2020
kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] deserialized teObject [TEObject 3, wrapping a payload of 249 bytes]
kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] payload: [DemoPayload, string=ghi, integer=3]

kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] **received consumerRecord** Wed Jul 08 20:25:14 MDT 2020
kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] deserialized teObject [TEObject 4, wrapping a payload of 249 bytes]
kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] payload: [DemoPayload, string=jkl, integer=4]

kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] **received consumerRecord** Wed Jul 08 20:25:14 MDT 2020
kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] deserialized teObject [TEObject 5, wrapping a payload of 249 bytes]
kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] payload: [DemoPayload, string=mno, integer=5]

The kafkaConsumer process thread is done reading tamper-evident records from the blockchain and shuts down.

kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] consumer loop poll...
kafkaConsumer [KafkaBlockchainMultiplePartitionDemo$ConsumerLoop] closing the consumer loop...

2. Describe blockchain partitions for the Kafka group demo-multiple-partition-blockchain-consumers

Consumer group 'demo-multiple-partition-blockchain-consumers' has no active members.

This step uses a Kafka built in utility to describe the kafka-demo-blockchain statistics for the five partitions.

```
GROUP                          TOPIC                          PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG
CONSUMER-ID     HOST            CLIENT-ID
demo-multiple-partition-blockchain-consumers kafka-demo-multiple-partition-blockchain 1          4               4
0               -               -               -
demo-multiple-partition-blockchain-consumers kafka-demo-multiple-partition-blockchain 2          0               0
0               -               -               -
demo-multiple-partition-blockchain-consumers kafka-demo-multiple-partition-blockchain 0          0               0
0               -               -               -
demo-multiple-partition-blockchain-consumers kafka-demo-multiple-partition-blockchain 3          1               1
0               -               -               -
demo-multiple-partition-blockchain-consumers kafka-demo-multiple-partition-blockchain 4          0               0
0               -               -               -
```

3. Demonstrate verifying messages from the Kafka blockchain named kafka-demo-multiple-partition-blockchain, having five partitions

This demonstration retrieves the genesis hash from ZooKeeper, then reads all the blockchain tamper-evident records from the blockchain topic, and compares their expected SHA-256 hashes with the recomputed hashes in sequence.

The main process thread connects to ZooKeeper and retrieves the genesis SHA-256 hash for the demo blockchain topic.

```
main [KafkaBlockchainMultiplePartitionDemoVerification] Verifying the specified Kafka blockchain named
kafka-demo-multiple-partition-blockchain
main [KafkaBlockchainMultiplePartitionDemoVerification] connecting to ZooKeeper...
main [KafkaBlockchainMultiplePartitionDemoVerification] Kafka topics [kafka-demo-blockchain,
kafka-demo-multiple-partition-blockchain]
main [KafkaBlockchainMultiplePartitionDemoVerification] genesis hash for path
/KafkaBlockchain/kafka-demo-multiple-partition-blockchain =
3e9801a24dbe3f19e0ae9430dd9dd942cb39f9b1431c25a56c63d8a0c8b0661e
main [KafkaUtils] getNbrPartitionsForTopic, topic: kafka-demo-multiple-partition-blockchain
```

The main process thread determines that the blockchain has five partitions and starts a consumer process thread for each one.

```
main [KafkaBlockchainMultiplePartitionDemoVerification] the blockchain named
kafka-demo-multiple-partition-blockchain has 5 partitions
main [KafkaBlockchainMultiplePartitionDemoVerification] starting consumer 0
main [KafkaBlockchainMultiplePartitionDemoVerification] starting consumer 1
main [KafkaBlockchainMultiplePartitionDemoVerification] starting consumer 2
main [KafkaBlockchainMultiplePartitionDemoVerification] starting consumer 3
```

main [KafkaBlockchainMultiplePartitionDemoVerification] starting consumer 4

Each of the five consumer process threads assigns itself to a corresponding topic partition.

consumer-3 [KafkaBlockchainMultiplePartitionDemoVerification] thread 3 **assigning topic partition** [kafka-demo-multiple-partition-blockchain-3]
consumer-1 [KafkaBlockchainMultiplePartitionDemoVerification] thread 1 assigning topic partition [kafka-demo-multiple-partition-blockchain-1]
consumer-0 [KafkaBlockchainMultiplePartitionDemoVerification] thread 0 assigning topic partition [kafka-demo-multiple-partition-blockchain-0]
consumer-4 [KafkaBlockchainMultiplePartitionDemoVerification] thread 4 assigning topic partition [kafka-demo-multiple-partition-blockchain-4]
consumer-2 [KafkaBlockchainMultiplePartitionDemoVerification] thread 2 assigning topic partition [kafka-demo-multiple-partition-blockchain-2]

Each of the five consumer process threads seeks to the beginning of their respective topic partition.

consumer-3 [KafkaBlockchainMultiplePartitionDemoVerification] thread 3 **seeking to the beginning of topic partition** [kafka-demo-multiple-partition-blockchain-3]
consumer-4 [KafkaBlockchainMultiplePartitionDemoVerification] thread 4 seeking to the beginning of topic partition [kafka-demo-multiple-partition-blockchain-4]
consumer-2 [KafkaBlockchainMultiplePartitionDemoVerification] thread 2 seeking to the beginning of topic partition [kafka-demo-multiple-partition-blockchain-2]
consumer-1 [KafkaBlockchainMultiplePartitionDemoVerification] thread 1 seeking to the beginning of topic partition [kafka-demo-multiple-partition-blockchain-1]
consumer-0 [KafkaBlockchainMultiplePartitionDemoVerification] thread 0 seeking to the beginning of topic partition [kafka-demo-multiple-partition-blockchain-0]

Each of the five consumer process threads polls their respective topic partition.

consumer-0 [KafkaBlockchainMultiplePartitionDemoVerification] thread 0 consumer poll...
consumer-2 [KafkaBlockchainMultiplePartitionDemoVerification] thread 2 consumer poll...
consumer-1 [KafkaBlockchainMultiplePartitionDemoVerification] thread 1 consumer poll...
consumer-4 [KafkaBlockchainMultiplePartitionDemoVerification] thread 4 consumer poll...
consumer-3 [KafkaBlockchainMultiplePartitionDemoVerification] thread 3 consumer poll...

Two of the five consumer process threads receive tamper-evident records.

consumer-1 [KafkaBlockchainMultiplePartitionDemoVerification] thread 1 **received 4 records** from partition [kafka-demo-multiple-partition-blockchain-1]
consumer-3 [KafkaBlockchainMultiplePartitionDemoVerification] thread 3 **received 1 records** from partition [kafka-demo-multiple-partition-blockchain-3]

consumer-3 [KafkaBlockchainMultiplePartitionDemoVerification] thread 3 deserialized teObject [**TEObject 1, wrapping a payload of 249 bytes**]

consumer-1 [KafkaBlockchainMultiplePartitionDemoVerification] thread 1 deserialized teObject [**TEObject 2, wrapping a payload of 249 bytes**]
consumer-0 [KafkaBlockchainMultiplePartitionDemoVerification] thread 0...end of consumed records
consumer-4 [KafkaBlockchainMultiplePartitionDemoVerification] thread 4...end of consumed records
consumer-0 [KafkaBlockchainMultiplePartitionDemoVerification] thread 0 closing the consumer
consumer-4 [KafkaBlockchainMultiplePartitionDemoVerification] thread 4 closing the consumer
consumer-2 [KafkaBlockchainMultiplePartitionDemoVerification] thread 2...end of consumed records
consumer-2 [KafkaBlockchainMultiplePartitionDemoVerification] thread 2 closing the consumer

The main process thread coordinates with the five consumer threads to find the first tamper-evident record in their respective partitions.

main [KafkaBlockchainMultiplePartitionDemoVerification] checking consumers for the first blockchain record in their respective partitions...
main [KafkaBlockchainMultiplePartitionDemoVerification] consumer 0, first record in partition [TEObject 2, wrapping a payload of 249 bytes]
main [KafkaBlockchainMultiplePartitionDemoVerification] consumer 1 is done
main [KafkaBlockchainMultiplePartitionDemoVerification] consumer 2 is done
main [KafkaBlockchainMultiplePartitionDemoVerification] consumer 3, first record in partition [TEObject 1, wrapping a payload of 249 bytes]
main [KafkaBlockchainMultiplePartitionDemoVerification] consumer 4 is done

The main process thread coordinates with the five consumer threads to find and to verify the tamper-evident record with serial number 1, which is the genesis record of the blockchain.

main [KafkaBlockchainMultiplePartitionDemoVerification] finding the genesis record for kafka-demo-multiple-partition-blockchain...
main [KafkaBlockchainMultiplePartitionDemoVerification] **found genesis record [TEObject 1, wrapping a payload of 249 bytes], having serialNbr 1**

The main process thread coordinates with the five consumer threads to find and to verify the tamper-evident record with successive serial numbers until the blockchain topic is exhausted.

consumer-3 [KafkaBlockchainMultiplePartitionDemoVerification] thread 3 consumer poll...
main [KafkaBlockchainMultiplePartitionDemoVerification] **found successor [TEObject 2, wrapping a payload of 249 bytes]** in partition 0
consumer-1 [KafkaBlockchainMultiplePartitionDemoVerification] thread 1 deserialized teObject [TEObject 3, wrapping a payload of 249 bytes]

main [KafkaBlockchainMultiplePartitionDemoVerification] **found successor [TEObject 3, wrapping a payload of 249 bytes]** in partition 0
consumer-1 [KafkaBlockchainMultiplePartitionDemoVerification] thread 1 deserialized teObject [TEObject 4, wrapping a payload of 249 bytes]

main [KafkaBlockchainMultiplePartitionDemoVerification] **found successor [TEObject 4, wrapping a payload of 249 bytes]** in partition 0

consumer-1 [KafkaBlockchainMultiplePartitionDemoVerification] thread 1 deserialized teObject [TEObject 5, wrapping a payload of 249 bytes]

main [KafkaBlockchainMultiplePartitionDemoVerification] found successor **[TEObject 5, wrapping a payload of 249 bytes]** in partition 0

The main process thread shuts down this demonstration.

consumer-1 [KafkaBlockchainMultiplePartitionDemoVerification] thread 1 consumer poll...
consumer-3 [KafkaBlockchainMultiplePartitionDemoVerification] thread 3...end of consumed records
consumer-3 [KafkaBlockchainMultiplePartitionDemoVerification] thread 3 closing the consumer
consumer-1 [KafkaBlockchainMultiplePartitionDemoVerification] thread 1...end of consumed records
consumer-1 [KafkaBlockchainMultiplePartitionDemoVerification] thread 1 closing the consumer
main [KafkaBlockchainMultiplePartitionDemoVerification] all blockchain partition consumers are done