

# KIP-664: Provide tooling to detect and abort hanging transactions


- [Status](#)
- [Motivation](#)
- [Proposed Changes](#)
  - [Detection](#)
  - [Analysis](#)
  - [Recovery](#)
- [Public Interfaces](#)
  - [DescribeProducers](#)
    - [Request Schema](#)
    - [Response Schema](#)
  - [ListTransactions](#)
    - [Request Schema](#)
    - [Response Schema](#)
  - [DescribeTransactions](#)
    - [Request Schema](#)
    - [Response Schema](#)
  - [WriteTxnMarkers](#)
    - [Request Schema](#)
    - [Response Schema](#)
  - [AdminClient APIs](#)
    - [Listing Transactions](#)
    - [Describing Transactions](#)
    - [Describing Producers](#)
    - [Aborting Transactions](#)
  - [Metrics](#)
  - [Command Line Tool](#)
    - [Listing Transactions](#)
    - [Finding Hanging Transactions](#)
    - [Describing Transactions](#)
    - [Describing Producers](#)
    - [Aborting Transactions](#)
- [Compatibility, Deprecation, and Migration Plan](#)
- [Rejected Alternatives](#)

## Status

**Current state:** Adopted

**Discussion thread:** <https://www.mail-archive.com/dev@kafka.apache.org/msg111066.html>

JIRA:

 Unable to render Jira issues macro, execution error.

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

## Motivation

In [KAFKA-9144](#), we encountered a bug which could lead to a transaction being left in a hanging state. When this happens, the last stable offset (LSO) is unable to advance which means that consumers using the `read_committed` isolation level will be stuck. Furthermore, if the topic is compacted, then the cleaner would be unable to clean beyond this offset which can lead to unbounded growth. Today there are no good recovery options for users who hit this problem. The only practical option is to wait for the data from the hanging transaction to get removed from the log once retention time has been reached. In the case of a compacted topic, options are even more scarce. Users in some cases have just deleted and recreated topics to get around this bug.

In this proposal, we aim to address this problem by providing the tools to recover from hanging transactions. As a part of this, we want to improve visibility into transactional state. Unlike the group coordinator, there are no APIs currently which let us introspect the state of the transaction coordinator or the producer state associated with each topic partition. Hence we want to add these APIs so that hanging transactions and any other EOS-related problems can be more easily diagnosed.

## Proposed Changes

This proposal provides the tooling support needed to detect, analyze, and safely recover from a hanging transaction.

## Detection

The first issue to address is how a user can find topic partitions that may have hanging transactions. Preferably we want a metric so that alerts can be triggered proactively rather than waiting for users to complain. Today, Kafka exposes a partition-level "LastStableOffsetLag" metric which indicates how far behind the LSO is from the log end offset. When there is a hanging transaction, the LSO lag will tend to grow indefinitely. However, it is difficult to assign an alert threshold because it depends on the characteristics of the application (e.g. transaction duration and throughput).

Ideally, we would prefer a metric which takes transaction timeout into account. Although partition leaders do not know the transaction timeout associated with each transactional write, they do know the upper bound indicated by `transaction.max.timeout.ms`. Any transaction which has remained open longer than this timeout ought to be treated as suspicious. We propose to add a new metric `PartitionsWithLateTransactionsCount`, which tracks the count of the number of partitions which have open transactions with durations exceeding `transaction.max.timeout.ms`. This gives users a simple alert criteria.

Note that it is possible to have transaction timeouts which are exactly equal to `transaction.max.timeout.ms`. To account for some extra latency and avoid spurious alerts, we will add 5 minutes of padding before a transaction is counted in `PartitionsWithLateTransactionsCount`.

## Analysis

Hanging transactions are the result of an inconsistent state between the replicas and the transaction coordinator. It is not easy to analyze a hanging transaction if one is expected today because there is little visibility into either the producer state maintained by each replica or the transaction state of the coordinator. We propose to add three new APIs to address this gap:

- **DescribeProducers**: this request is sent to partition leaders to describe the state of the active idempotent/transactional producers.
- **ListTransactions**: this request is sent to any broker to collect the current set of TransactionalId/ProducerId which have pending transactions.
- **DescribeTransactions**: this request is sent to transaction coordinators to describe the transaction state in more detail (including the topic partitions)

The schemas for these APIs are provided in the "Public Interfaces" section below. Combined, they give the user a way to inspect ongoing transaction state when a specific topic partition is suspected to have a hanging transaction.

At a high level, we suggest the following workflow once a topic partition with a suspected hanging transaction has been identified:

1. Use `DescribeProducers` to collect the set of ProducerIds which have transactions exceeding the max transaction timeout
2. Use `ListTransactions` to the available brokers to find the the TransactionalIds associated with these ProducerIds.
3. Finally, use `DescribeTransactions` to validate the transaction state and ensure it is safe to abort.

This proposal adds a command line tool which will automates this process of analysis.

## Recovery

The remaining problem to solve is how to safely abort a hanging transaction. We propose to extend the `WriteTxnMarker` API so that it can be used by the Kafka AdminClient. Currently we use the coordinator epoch (which is the leader epoch of the associated `__transaction_state` partition) as a kind of concurrency control. Basically partition leaders will not accept non-monotonic updates for a given `ProducerId`. We need to ensure that writes from the AdminClient do not interfere with this mechanism.

The approach we take here is to add the start offset of the transaction that is intended to be aborted. The partition leader will accept the request only if there is an open transaction beginning at the offset specified in the request. Additionally we require the producer epoch in the request to exactly match the latest value. This ensures that the transaction coordinator remains the only one that can bump the epoch of a transactional producer. So we couldn't hit a case where the epoch in some partition's state got ahead of the coordinator which might cause worse failures. Finally, markers written from the AdminClient will set `coordinator_epoch` in the request to -1. This avoids race conditions with concurrent requests from existing coordinators and gives us the ability to tell that the source of the marker was the AdminClient.

## Public Interfaces

### DescribeProducers

The `DescribeProducers` API will require `Read` permission on the associated topic. It is expected to be sent to any replica of a topic partition.

### Request Schema

This request is expected to be sent to partition leaders.

```
{
  "apiKey": 61,
  "type": "request",
  "name": "DescribeProducersRequest",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "Topics", "type": "[[]TopicRequestData", "versions": "0+", "fields": [
      { "name": "Name", "type": "string", "versions": "0+", "entityType": "topicName",
        "about": "The topic name." },
      { "name": "PartitionIndexes", "type": "[[]int32", "versions": "0+",
        "about": "The indexes of the partitions to list producers for." }
    ]}
  ]}
}
```

## Response Schema

```
{
  "apiKey": 61,
  "type": "response",
  "name": "DescribeProducersResponse",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "ThrottleTimeMs", "type": "int32", "versions": "0+", "ignorable": true,
      "about": "The duration in milliseconds for which the request was throttled due to a quota violation, or zero if the request did not violate any quota." },
    { "name": "Topics", "type": "[[]TopicResponse", "versions": "0+",
      "about": "Each topic in the response.", "fields": [
        { "name": "Name", "type": "string", "versions": "0+", "entityType": "topicName",
          "about": "The topic name" },
        { "name": "Partitions", "type": "[[]PartitionResponse", "versions": "0+",
          "about": "Each partition in the response.", "fields": [
            { "name": "PartitionIndex", "type": "int32", "versions": "0+",
              "about": "The partition index." },
            { "name": "ErrorCode", "type": "int16", "versions": "0+",
              "about": "The partition error code, or 0 if there was no error." },
            { "name": "ActiveProducers", "type": "[[]ProducerState", "versions": "0+", "fields": [
              { "name": "ProducerId", "type": "int64", "versions": "0+" },
              { "name": "ProducerEpoch", "type": "int32", "versions": "0+" },
              { "name": "LastSequence", "type": "int32", "versions": "0+" },
              { "name": "LastTimestamp", "type": "int64", "versions": "0+" },
              { "name": "TxnStartOffset", "type": "int64", "versions": "0+" },
              { "name": "CoordinatorEpoch", "type": "int32", "versions": "0+" },
            ]}
          ]}
        ]}
      ]}
    ]}
  ]}
}
```

### Possible errors:

- NOT\_LEADER\_OR\_FOLLOWER: If the replica receiving the request is not the current leader of a topic partition
- TOPIC\_AUTHORIZATION\_FAILED: If the user does not have Describe access on a requested topic.
- UNKNOWN\_TOPIC\_OR\_PARTITION: If either the topic or partition is not known to exist.

## ListTransactions

This API is analogous to `ListGroups`` for the group coordinator and requires `Describe`` permission on the `TransactionId`` resource. That is, it will only include transactions for which the principal has `Describe`` permission on the corresponding `TransactionalId``.

It is meant to provide a summary of the transactions managed by a given coordinator.

## Request Schema

The request includes a filter of the states that the use is interested in. As an example, this allows us to filter only the "Ongoing" transactions. We have also included a filter for the `ProducerId`. This is useful when trying to reverse lookup the `TransactionalId` as we need to do to support the `--find-hanging` command below.

```
{
  "apiKey": 66,
  "type": "request",
  "name": "ListTransactionsRequest",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "StatesFilter", "type": "[string", "versions": "0+",
      "about": "The states of transactions we want to list."
    },
    { "name": "ProducerIdFilter", "type": "[int64", "versions": "0+",
      "about": "Array of ProducerIds to limit the response to"
    }
  ]
}
```

## Response Schema

```
{
  "apiKey": 66,
  "type": "response",
  "name": "ListTransactionsResponse",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "ThrottleTimeMs", "type": "int32", "versions": "0+", "ignorable": true,
      "about": "The duration in milliseconds for which the request was throttled due to a quota violation, or zero if the request did not violate any quota." },
    { "name": "ErrorCode", "type": "int16", "versions": "0+" },
    { "name": "TransactionStates", "type": "[TransactionState", "versions": "0+", "fields": [
      { "name": "TransactionalId", "type": "string", "versions": "0+" },
      { "name": "ProducerId", "type": "int64", "versions": "0+" },
      { "name": "TransactionState", "type": "string", "versions": "0+" }
    ]}
  ]
}
```

### Possible Error Codes:

- `COORDINATOR_LOAD_IN_PROGRESS`: The coordinator is in the process of loading its state.
- `COORDINATOR_NOT_AVAILABLE`: If the coordinator receiving the request is being shutdown.

## DescribeTransactions

The `DescribeTransactions` API will require `Describe` permission on the associated `TransactionalId` resource.

## Request Schema

The `DescribeTransactions` request is sent to transaction coordinators. Similarly to the `DescribeGroups` API, we expect the `AdminClient` to use `FindCoordinator` first in order to find the right coordinator when a specific `transactionalId` is requested. Note that the array of transactional IDs is not nullable. The API does not provide a way to list the states of all `TransactionalIds` on a given broker.

The request schema is specified below:

```
{
  "apiKey": 65,
  "type": "request",
  "name": "DescribeTransactionsRequest",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "TransactionalIds", "type": "[]string", "versions": "0+" }
  ]
}
```

## Response Schema

The response schema is specified below:

```
{
  "apiKey": 65,
  "type": "response",
  "name": "DescribeTransactionsResponse",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "ThrottleTimeMs", "type": "int32", "versions": "0+", "ignorable": true,
      "about": "The duration in milliseconds for which the request was throttled due to a quota violation, or zero if the request did not violate any quota." },
    { "name": "TransactionStates", "type": "[]TransactionState", "versions": "0+", "fields": [
      { "name": "ErrorCode", "type": "int16", "versions": "0+" },
      { "name": "TransactionalId", "type": "string", "versions": "0+" },
      { "name": "TransactionState", "type": "int8", "versions": "0+" },
      { "name": "TransactionTimeoutMs", "type": "int32", "versions": "0+" },
      { "name": "TransactionStartTimeMs", "type": "int64", "versions": "0+" },
      { "name": "ProducerId", "type": "int64", "versions": "0+" },
      { "name": "ProducerEpoch", "type": "int32", "versions": "0+" },
      { "name": "TopicPartitions", "type": "[]TopicData", "versions": "0+", "fields": [
        { "name": "PartitionIndexes", "type": "[]int32", "versions": "0+" }
      ]
    }
  ]
}
```

These fields are derived from the transaction log itself. The API is mainly just providing a more convenient way to access the state than dumping the log.

The transaction state is the same field included in the `ListTransactions` API.

When there is no transaction in progress, the `TransactionStartTimeMs` field will be set to -1 and the set of topic partitions will be empty. Additionally we will reserve the status code of -1 to handle states which are unknown to the client.

### Possible Error Codes:

- NOT\_COORDINATOR: If the coordinator receiving the request does not own a transactionalId in the request.
- COORDINATOR\_LOAD\_IN\_PROGRESS: The coordinator is in the process of loading its state.
- COORDINATOR\_NOT\_AVAILABLE: If the coordinator receiving the request is being shutdown.
- TRANSACTIONAL\_ID\_NOT\_FOUND (NEW): New error code which indicates that the requested TransactionalId could not be found
- TRANSACTIONAL\_ID\_AUTHORIZATION\_FAILED: If the principal does not have Describe permission one of the transactionalIds in the request.

## WriteTxnMarkers

As described above, we are bumping the `WriteTxnMarkers` API so that we can include the start offset of the transaction that should be administratively aborted. Transaction coordinators will leave this field unspecified.

Note that we are not changing the required permission for this API. It will still request `ClusterAction`: only administrators should be allowed to abort a transaction manually.

## Request Schema

Below we define the changes to the request schema:

```

{
  "apiKey": 27,
  "type": "request",
  "name": "WriteTxnMarkersRequest",
  "validVersions": "0-1",
  "flexibleVersions": "1+",
  "fields": [
    { "name": "Markers", "type": "[]WritableTxnMarker", "versions": "0+",
      "about": "The transaction markers to be written.", "fields": [
        { "name": "ProducerId", "type": "int64", "versions": "0+", "entityType": "producerId",
          "about": "The current producer ID." },
        { "name": "ProducerEpoch", "type": "int16", "versions": "0+",
          "about": "The current epoch associated with the producer ID." },
        { "name": "TransactionResult", "type": "bool", "versions": "0+",
          "about": "The result of the transaction to write to the partitions (false = ABORT, true = COMMIT)." },
        { "name": "Topics", "type": "[]RequestTopic", "versions": "0+",
          "about": "Each topic that we want to write transaction marker(s) for.", "fields": [
            { "name": "Name", "type": "string", "versions": "0+", "entityType": "topicName",
              "about": "The topic name." },
            { "name": "PartitionIndexes", "type": "[]int32", "versions": "0",
              "about": "The indexes of the partitions to write transaction markers for." }
          ]
        },
        // NEW: The array below replaces "PartitionIndexes" and adds "TxnStartOffset"
        // as an optional field, which will only be included by the AdminClient
        { "name": "Partitions", "type": "[]RequestPartition", "versions": "1+", "fields": [
          { "name": "PartitionIndex", "type": "int32", "versions": "1+",
            "about": "The partition index." },
          { "name": "TxnStartOffset", "type": "int64", "versions": "1+", "taggedVersions": "1+", "tag": 0,
            "about": "The start offset of the transaction that should be completed" }
        ]
      }
    ],
    { "name": "CoordinatorEpoch", "type": "int32", "versions": "0+",
      "about": "Epoch associated with the transaction state partition hosted by this transaction coordinator"
    }
  ]
}

```

## Response Schema

There are no changes to the response schema, but it will be bumped. Note that we are also enabling flexible version support. Note also that `INVALID_TXN_STATE` is now a possible error code when the `TxnStartOffset` is included in the request.

```

{
  "apiKey": 27,
  "type": "response",
  "name": "WriteTxnMarkersResponse",
  "validVersions": "0-1",
  "flexibleVersions": "1+",
  "fields": [
    {
      "name": "Markers", "type": "[]WritableTxnMarkerResult", "versions": "0+",
      "about": "The results for writing makers.", "fields": [
        {
          "name": "ProducerId", "type": "int64", "versions": "0+", "entityType": "producerId",
          "about": "The current producer ID in use by the transactional ID." },
        {
          "name": "Topics", "type": "[]TopicResult", "versions": "0+",
          "about": "The results by topic.", "fields": [
            {
              "name": "Name", "type": "string", "versions": "0+", "entityType": "topicName",
              "about": "The topic name." },
            {
              "name": "Partitions", "type": "[]PartitionResult", "versions": "0+",
              "about": "The results by partition.", "fields": [
                {
                  "name": "PartitionIndex", "type": "int32", "versions": "0+",
                  "about": "The partition index." },
                {
                  "name": "ErrorCode", "type": "int16", "versions": "0+",
                  "about": "The error code, or 0 if there was no error." }
              ]
            }
          ]
        }
      ]
    }
  ]
}

```

#### Possible Error Codes:

- **NOT\_LEADER\_OR\_FOLLOWER**: If the replica receiving the request is not the current leader of a topic partition
- **CLUSTER\_AUTHORIZATION\_FAILED**: If the user does not have `ClusterAction` permission.
- **UNKNOWN\_TOPIC\_OR\_PARTITION**: If either the topic or partition is not known to exist.
- **INVALID\_PRODUCER\_EPOCH**: If no cluster epoch is provided and the provided epoch does not exactly match the current
- **INVALID\_TXN\_STATE (NEW)**: If there is no transaction in progress at the indicated offset

## AdminClient APIs

The new APIs will be exposed from the `AdminClient` following the usual conventions:

```

interface Admin {
  ListTransactionsResult listTransactions();
  ListTransactionsResult listTransactions(ListTransactionsOptions options);

  DescribeTransactionsResult describeTransactions(Collection<String> transactionalIds);
  DescribeTransactionsResult describeTransactions(Collection<String> transactionalIds,
  DescribeTransactionsOptions options);

  DescribeProducersResult describeProducers(Collection<TopicPartition> partitions);
  DescribeProducersResult describeProducers(Collection<TopicPartition> partitions, DescribeProducersOptions
  options);

  AbortTransactionResult abortTransaction(TransactionInfo info);
  AbortTransactionResult abortTransaction(TransactionInfo info, AbortTransactionOptions options);
}

```

The sections below describe the supporting classes in more detail.

## Listing Transactions

```

class ListTransactionsOptions extends AbstractOptions<ListTransactionsOptions> {
    ListTransactionsOptions setBrokerId(int brokerId);
    OptionalInt brokerId();
}

class DescribeTransactionsResult {
    KafkaFuture<Map<Integer, Collection<TransactionSummary>>> all();
    KafkaFuture<Collection<TransactionSummary>>> brokerResult(Integer brokerId);

    static class TransactionSummary {
        final String transactionalId;
        final long producerId;
        final String transactionState;
    }
}

```

## Describing Transactions

```

class DescribeTransactionsOptions extends AbstractOptions<DescribeTransactionsOptions> {
}

class DescribeTransactionsResult {
    KafkaFuture<Map<String, TransactionState>>> all();
    KafkaFuture<TransactionState> partitionResult(String transactionalId);

    static class TransactionState {
        final String transactionState;
        final int transactionTimeoutMs;
        final long transactionStartTimeMs;
        final long producerId;
        final int producerEpoch;
        final List<TopicPartition> addedPartitions;
    }
}

```

## Describing Producers

```

class DescribeProducersOptions extends AbstractOptions<DescribeProducersOptions> {
    DescribeProducersOptions setBrokerId(int brokerId);
    OptionalInt brokerId();
}

class DescribeProducersResult {
    KafkaFuture<Map<TopicPartition, PartitionProducerState>>> all();
    KafkaFuture<PartitionProducerState> partitionResult(TopicPartition partition);

    static class PartitionProducerState {
        final List<ProducerState> activeProducers;
    }

    static class ProducerState {
        final long producerId;
        final int producerEpoch;
        final int lastSequence;
        final long lastTimestamp;
        final OptionalLong currentTransactionStartOffset;
        final int coordinatorEpoch;
    }
}

```

## Aborting Transactions

```

class AbortTransactionOptions extends AbstractOptions<AbortTransactionOptions> {
}

abstract class TransactionInfo {
    final TopicPartition topicPartition;
    final long ProducerId;
    final int producerEpoch;

    // One and only one of the following fields must be set
    final OptionalLong transactionStartOffset;
    final OptionalInt coordinatorEpoch;
}

class AbortTransactionResult {
    KafkaFuture<Void> result();
}

```

## Metrics

As discussed above, we will add new gauge `PartitionsWithLateTransactionsCount`, which is tracked in the `ReplicaManager` group (along with `UnderMinIsrPartitionCount`, `OfflineReplicaCount`, etc.). This metric will record the number of partitions which have open transactions with durations exceeding `transaction.max.timeout.ms` (plus 5 minutes).

## Command Line Tool

There will be a new tool: `kafka-transactions.sh`. The following commands will be supported:

- `--list`: List transactions known to a transaction coordinator
- `--find-hanging`: Find hanging transactions on a specific broker
- `--describe`: Describe details specific to a particular transactionalId
- `--describe-producers`: Describe active producer state for a given topic partition
- `--abort`: Forcefully abort a transaction

## Listing Transactions

The `--list` command allows a user to list transaction state from the transaction coordinators. It supports an optional `--broker` flag to select a specific node.

```

> kafka-transactions.sh --list --bootstrap-server localhost:9092
TransactionalId      ProducerId      Coordinator State
my-txn-id1           134132         0              Ongoing
my-txn-id2           134147         0              Ongoing
my-txn-id3           134191         1              PrepareCommit
my-txn-id4           134193         2              CompleteAbort

> kafka-transactions.sh --list --broker 0 --bootstrap-server localhost:9092
TransactionalId      ProducerId      Coordinator State
my-txn-id1           134132         0              Ongoing
my-txn-id2           134147         0              Ongoing

```

## Finding Hanging Transactions

The `--find-hanging` command is used to automate the process of finding hanging transactions on a specific broker. It has one required argument `--max-transaction-timeout`. Internally what this will do is the following:

1. Use the `Metadata` API to find all of the partitions.
2. Send `DescribeProducers` including each of the partitions found in the first step.
3. Collect any open transactions which have been open longer than the provided `--max-transaction-timeout`.
4. Use `ListTransactions` on each available coordinator to find the respective `TransactionalId` and coordinator for each `ProducerId` found in the step above.
  - a. If no coordinator could be found, then the transaction is considered hanging.
  - b. Otherwise, use `DescribeTransactions` to determine the state of the transaction and decide whether it should be considered hanging (e.g. if the epoch does not match or the partition is not included in the current transaction).

Here is an example:

```
> kafka-transactions.sh --find-hanging --broker 0 --bootstrap-server localhost:9092
Topic          Partition      ProducerId      ProducerEpoch      StartOffset
LastTimestamp      Duration(s)
foo              0              134132          23
550              2020-09-17T23:02:23Z          30

# Limit the search to a specific topic partition
> kafka-transactions.sh --find-hanging --broker 0 --topic foo --partition 0 --bootstrap-server localhost:9092
Topic          Partition      ProducerId      ProducerEpoch      StartOffset
LastTimestamp      Duration(s)
foo              0              134132          23
550              2020-09-17T23:02:23Z          30000
```

## Describing Transactions

The `--describe` command can be used to describe the state of a transaction when the `TransactionalId` is known. It uses the `'DescribeTransactions'` API.

```
# Describe transaction state for my-txn-id
> kafka-transaction.sh --describe --transactional-id my-txn-id --bootstrap-server localhost:9092
ProducerId ProducerEpoch Coordinator State      TimeoutMs TopicPartitions
134132      24              0          Ongoing 5000      foo-0,foo-1
```

## Describing Producers

The `--describe-producers` command is used to describe the producer state of a specific topic partition. Internally, it is a direct call to the `'DescribeProducers'` API. It takes an optional `--broker` argument to specify a specific replica.

```
# Describe producers on the leader of a topic partition
> kafka-transaction.sh --describe-producers --topic foo --partition 0 --bootstrap-server localhost:9092
ProducerId      ProducerEpoch      StartOffset      LastTimestamp      Duration
(s)      CoordinatorEpoch
134132          23              77              550              2020-09-17T23:02:
23Z          30
134938          5              439              2020-09-17T23:01:
23Z          90              64

# Describe producers for a specific replica of a topic partition
> kafka-transaction.sh --describe-producers --broker 0 --topic foo --partition 0 --bootstrap-server localhost:
9092
ProducerId      ProducerEpoch      StartOffset      LastTimestamp      Duration
(s)      CoordinatorEpoch
134132          23              77              550              2020-09-17T23:02:
23Z          30
134938          5              439              2020-09-17T23:01:
23Z          90              64
```

## Aborting Transactions

The `--abort` command can be used to abort an ongoing transaction for a topic partition. It has several required arguments:

- `--topic`
- `--partition`
- `--start-offset`

Internally, the tool will first use `'DescribeProducers'` to validate that there is an open transaction beginning at that offset and collect the `ProducerId` and `ProducerEpoch`.

For compatibility with older brokers, we also support the ability to directly specify additional parameters. Although we cannot use the new APIs in this case to validate the state, this is still better than nothing for users who have hit this problem. In this case, users have no choice but to do the analysis themselves by dumping the log of a suspected topic partition. In this case, the command takes the following arguments:

- `--topic`
- `--partition`
- `--producer-id`
- `--producer-epoch`

- `--coordinator-epoch`

As described above, the partition leader will allow an administrative abort only if the producer epoch matches the latest value and there is an open transaction beginning at the indicated offset. Users are required to use the `--describe` command to find this information.

```
> kafka-transactions.sh --abort \  
  --topic foo \  
  --partition 0 \  
  --start-offset 550 \  
  --bootstrap-server localhost:9092  
  
> kafka-transactions.sh --abort \  
  --topic foo \  
  --partition 0 \  
  --producer-id 134132 \  
  --producer-epoch 23 \  
  --coordinator-epoch 15 \  
  --bootstrap-server localhost:9092
```

Note that we are not providing an option to commit a transaction through this API.

## Compatibility, Deprecation, and Migration Plan

This proposal adds new tools to facilitate the recovery of hanging transactions. These tools will not generally be compatible with older versions since they rely on new APIs. However, the `--abort` command described above will be compatible with older brokers.

## Rejected Alternatives

If there were a simple and safe way to automatically detect which transactions were left hanging, then an automatic recovery option might be preferable. We have considered two options:

**Transaction State Reconciliation:** A hanging transaction is the result of an inconsistent state between the transaction coordinator and the individual topic partitions which were included in a transaction. The replicas of a topic partition maintain some state about the ongoing transactions which affect that topic partition. They know which transactions are *open*, but not necessarily which ones have been left hanging. In order to detect the hanging transactions, there needs to be some way to reconcile the state of the transaction coordinator.

Unfortunately, there is a challenge here. The transaction coordinator mapping is done using the `transactionalId` of the producer, but the replicas of a topic partition only see the `producerId`. There is no way to reverse map from the `producerId` to the `transactionalId`, which means there is also no way for a replica to find the respective transaction coordinator. This means that reconciling the state of an open transaction requires querying *all* transaction coordinators, which suggests an expensive process and a lot of bookkeeping.

**Transaction Timeout:** Another way to detect a hanging transaction is through the transaction timeout. Although a replica has no way to know the precise transaction timeout that is used by a producer, it does know the upper bound as indicated through `transaction.max.timeout.ms`. Any transaction which has been left open longer than this has potentially been left hanging by the transaction coordinator. The main problem is ruling out false positives. Another explanation for a transaction which has been left open is that the coordinator has been unable to send the `WriteTxnMarker` request (e.g. because of a network partition). If we abort the transaction without verifying the state with the transaction coordinator, then we may violate the transactional guarantee.

Again, we emphasize that hanging transactions are not an expected state of the system, but the result of a bug. Even if automatic recovery could be done safely, there is a risk that it would cause the underlying bug to go unnoticed.