

KIP-859: Add Metadata Log Processing Error Related Metrics


- [Status](#)
- [Motivation](#)
- [Public Interfaces](#)
- [Proposed Changes](#)
 - [Controllers](#)
 - [Brokers](#)
- [Compatibility, Deprecation, and Migration Plan](#)
- [Rejected Alternatives](#)

Status

Current state: Accepted

Discussion thread: <https://lists.apache.org/thread/yl87h1s484yc09yjo1no46hwpbv0qkwt>

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

[KIP-500: Replace ZooKeeper with a Self-Managed Metadata Quorum](#) changes the way cluster metadata is stored and managed in a Kafka cluster. It introduces the concept of a replicated log that is maintained using a custom version of the Raft consensus protocol described in [KIP-595: A Raft Protocol for the Metadata Quorum](#). The controller now utilizes this log to persist and broadcast all metadata related actions in the cluster as described in [KIP-631: The Quorum-based Kafka Controller](#).

With these changes in place, the replicated log containing all metadata changes (henceforth called metadata log) is the source of metadata related information for all nodes in the cluster. Any errors that occur while processing the log could lead to the in-memory state for the node becoming inconsistent. It is important that such errors are made visible. The metrics proposed in the following section aim at doing so. These metrics can be used to set up alerts so that affected nodes can be discovered and needed remedial actions can be performed on them.

Public Interfaces

We propose adding the following new metrics:

Name	Description
<code>kafka.server:type=broker-metadata-metrics,name=metadata-apply-error-count</code>	Reports the number of errors encountered by the <code>BrokerMetadataPublisher</code> while applying a new <code>MetadataImage</code> based on the latest <code>MetadataDelta</code> .
<code>kafka.server:type=broker-metadata-metrics,name=metadata-load-error-count</code>	Reports the number of errors encountered by the <code>BrokerMetadataListener</code> while loading the metadata log and generating a new <code>MetadataDelta</code> based on it.
<code>kafka.controller:type=KafkaController,name=MetadataErrorCount</code>	Reports the number of times this controller node has encountered an error during metadata log processing

Proposed Changes

Controllers

The `MetadataErrorCount` metric is update for both active and standby controllers. For Active Controllers it is incremented anytime they hit an error in either generating a Metadata log or while applying it to memory. For standby controllers, this metric is incremented when they hit an error in applying the metadata log to memory. This metric will reflect the total count of errors that a controller encountered in metadata log processing since the last restart.

<https://github.com/apache/kafka/blob/14d2269471141067dc3c45300187f20a0a051777/metadata/src/main/java/org/apache/kafka/controller/QuorumController.java#L409>

handleEventException

```
private Throwable handleEventException(String name,
                                       OptionalLong startProcessingTimeNs,
                                       Throwable exception) {
    if (!startProcessingTimeNs.isPresent()) {
        ...
        ...
        renounce();
        //**** Increment MetadataErrorCount
        return new UnknownServerException(exception);
    }
}
```

Brokers

The metadata-apply-error-count metric will be incremented by one every time there is an error in publishing a new MetadataImage. This metric will reflect the count of cumulative errors since the broker started up.

<https://github.com/apache/kafka/blob/14d2269471141067dc3c45300187f20a0a051777/core/src/main/scala/kafka/server/metadata/BrokerMetadataPublisher.scala#L125>

Publish

```
override def publish(delta: MetadataDelta, newImage: MetadataImage): Unit = {
    val highestOffsetAndEpoch = newImage.highestOffsetAndEpoch()

    try {
        trace(s"Publishing delta $delta with highest offset $highestOffsetAndEpoch")

        // Publish the new metadata image to the metadata cache.
        metadataCache.setImage(newImage)
        ...
        ...
        publishedOffsetAtomic.set(newImage.highestOffsetAndEpoch().offset)
    } catch {
        //**** Increment metadata-apply-error-count
        case t: Throwable => error(s"Error publishing broker metadata at $highestOffsetAndEpoch", t)
        throw t
    } finally {
        _firstPublish = false
    }
}
```

The metadata-load-error-count metric will be incremented every time there is an error in loading batches and generating MetadataDelta from them. This metric will reflect the count of cumulative errors since the broker started up.

<https://github.com/apache/kafka/blob/14d2269471141067dc3c45300187f20a0a051777/core/src/main/scala/kafka/server/metadata/BrokerMetadataListener.scala#L112>

HandleCommitsEvent

```
class HandleCommitsEvent(reader: BatchReader[ApiMessageAndVersion])
  extends EventQueue.FailureLoggingEvent(log) {
  override def run(): Unit = {
    val results = try {
      val loadResults = loadBatches(_delta, reader, None, None, None)
      ...
      loadResults
    } catch {
      //**** Increment metadata-load-error-count
    } finally {
      reader.close()
    }

    ...
    _publisher.foreach(publish)
  }
}
```

<https://github.com/apache/kafka/blob/14d2269471141067dc3c45300187f20a0a051777/core/src/main/scala/kafka/server/metadata/BrokerMetadataListener.scala#L162>

HandleSnapshotEvent

```
class HandleSnapshotEvent(reader: SnapshotReader[ApiMessageAndVersion])
  extends EventQueue.FailureLoggingEvent(log) {
  override def run(): Unit = {
    try {
      info(s"Loading snapshot ${reader.snapshotId().offset}-${reader.snapshotId().epoch}.")
      _delta = new MetadataDelta(_image) // Discard any previous deltas.
      val loadResults = loadBatches(
        ...
      ) catch {
        //**** Increment metadata-load-error-count
      } finally {
        reader.close()
      }
      _publisher.foreach(publish)
    }
  }
}
```

Compatibility, Deprecation, and Migration Plan

These will be newly exposed metrics and there will be no impact on existing kafka versions.

Rejected Alternatives

Instead of adding the specific metrics, we could have added a more generic `MetadataProcessingErrorCount` Metric which would be incremented when either of these (and any other similar) or any other similar errors are hit on both Brokers and Controllers. The downside to this approach would be the loss in granularity on what exactly failed on a given node. The specific metrics are more meaningful and give better control over any alarming that might be setup on these metrics.