# KIP-989: RocksDB Iterator Metrics



## Status

**Current state**: Under Discussion

**Discussion thread**:

**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

RocksDB `Iterators` must be closed after use, to prevent memory leaks due to blocks being "pinned" in-memory. Pinned blocks can currently be tracked via the per-store `block-cache-pinned-usage` metric. However, it's common (and even recommended) to share the Block Cache among all stores in an application, to enable users to globally bound native memory used by RocksDB. This results in the `block-cache-pinned-usage` reporting the same memory usage for every store in the application, irrespective of which store is actually pinning blocks in the block cache.

To aid users in finding leaked Iterators, as well as identifying the cause of a high number of pinned blocks, we introduce two new metrics.

## Public Interfaces

New Metrics

| Name | Type | Scope | Description |
|---|---|---|---|
| `number-open-iterators` | Value | `stream-state-metrics` | The current number of Iterators on the store that have been created, but not yet closed. |
| `iterator-duration-avg` | Average | `stream-state-metrics` | The average time spent between creating an Iterator and closing it, in milliseconds. |
| `iterator-duration-total` | Total | `stream-state-metrics` | The total time spent between creating Iterators and closing them, in milliseconds. |

## Proposed Changes

The `number-open-iterators` metric will periodically record the difference between the `NO_ITERATOR_CREATED` and `NO_ITERATOR_DELETED` RocksDB Tickers. This value is likely to be non-zero in any application that heavily uses Iterators, even if they're being used properly and are not a performance concern. Users would want to monitor this metric and take action if:

- It is continually increasing, which would indicate an "Iterator leak"; i.e. one or more Iterators where `close()` is not being called.
- It is consistently very high, which would indicate a large number of concurrently open Iterators. This can be an indicator that some Iterators are taking a long time to complete processing, perhaps because the code being executed within each iteration performs poorly.

The `iterator-duration` metrics will be updated whenever an Iterator's close() method is called, recording the time since the Iterator was created. The measurement will be conducted using `System.nanoTime()` and reported in milliseconds, with a fractional component approximating the sub-millisecond precision. e.g. `2.337583`, would be approximately 2 milliseconds, 337 microseconds and 583 nanoseconds. Users would want to monitor this metric and take action if:

- The `avg`, or rate of change of the `total`, is consistently high, or continues to climb indefinitely. This would indicate a performance problem with code executed within each iteration of an Iterator.
- The metric reports 0, or no data, despite the application making use of RocksDB Iterators. This can indicate that the Iterators being used are not having their `close()` method called by the user, which would cause a memory leak.

## Compatibility, Deprecation, and Migration Plan

- No impact is expected for existing users, as no existing interfaces or behaviour is being changed.
- The performance overheads of tracking the opening/closing of Iterators is expected to be negligible to the point that it will not be measurable in real-world applications.

## Test Plan

RocksDBMetricsRecorderTest will be updated to include the new metrics.

## Rejected Alternatives

No alternatives were presented or considered.