KIP-836: Expose replication information of the cluster metadata

Status

Current state:"Approved"

Discussion thread: here

JIRA: KAFKA-13888

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

Motivation

The `DescribeQuorum` API as defined in KIP-595 is intended to allow the admin client to query the status of the Kraft quorum, including information about voter lag.

At this moment, if though it is a public API, the `DescribeQuorum` API is not accessible via the admin client. Furthermore, as implemented, the API response reports the state of the voters in terms of their `LogEndOffsets`. While useful, this information by itself is not an accurate measure of voter lag. T his information gives us some hint about what the voters' state but it is not a complete check as there is no good way to define an upper bound on how much lag in the `LogEndOffset` could be problematic. The divergence in this value is dependent upon the metadata load on the cluster at the time of measurement.

This KIP proposes making `DescribeQuorum` API accessible via the admin client and augmenting the `DescribeQuorum` API response with more information to be able to ascertain the **liveness** and **lag** of the voters in the quorum more accurately.

Public Interfaces

Additional Classes to expose the DescribeMetadataQuorumResult API to the admin client:

```
public class DescribeMetadataQuorumResult {

    private final KafkaFuture<QuorumInfo> quorumInfo;

    DescribeMetadataQuorumResult(KafkaFuture<QuorumInfo> quorumInfo) {
        this.quorumInfo = quorumInfo;
    }

    /**
     * Returns a future containing the QuorumInfo
     */
    public KafkaFuture<QuorumInfo> quorumInfo() {
        return quorumInfo;
    }
}
```

```
public class QuorumInfo {
    private final Integer leaderId;
```

```
private final List<ReplicaState> voters;
private final List<ReplicaState> observers;

QuorumInfo(Integer leaderId, List<ReplicaState> voters, List<ReplicaState> observers) {
    this.leaderId = leaderId;
    this.voters = voters;
    this.observers = observers;
}

public Integer leaderId() {
    return leaderId;
}

public List<ReplicaState> voters() {
    return voters;
}

public List<ReplicaState> observers() {
    return observers;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    QuorumInfo that = (QuorumInfo) o;
    return leaderId.equals(that.leaderId)
        && voters.equals(that.voters)
        && observers.equals(that.observers);
}

@Override
public int hashCode() {
    return Objects.hash(leaderId, voters, observers);
}

@Override
public String toString() {
    return "QuorumInfo(" +
        "leaderId=" + leaderId +
        ", voters=" + voters +
        ", observers=" + observers +
        ')';
}

public static class ReplicaState {
    private final int replicaId;
    private final long logEndOffset;
    private final OptionalLong lastFetchTimestamp;
    private final OptionalLong lastCaughtUpTimestamp;

    ReplicaState() {
        this(0, 0, OptionalLong.empty(), OptionalLong.empty());
    }

    ReplicaState(
        int replicaId,
        long logEndOffset,
        OptionalLong lastFetchTimestamp,
        OptionalLong lastCaughtUpTimestamp
    ) {
        this.replicaId = replicaId;
        this.logEndOffset = logEndOffset;
        this.lastFetchTimestamp = lastFetchTimestamp;
        this.lastCaughtUpTimestamp = lastCaughtUpTimestamp;
    }

    /**
     * Return the ID for this replica.
     * @return The ID for this replica
     */
```

```
        public int replicaId() {
            return replicaId;
        }

        /**
         * Return the logEndOffset known by the leader for this replica.
         * @return The logEndOffset for this replica
         */
        public long logEndOffset() {
            return logEndOffset;
        }

        /**
         * Return the lastFetchTime in milliseconds for this replica.
         * @return The value of the lastFetchTime if known, empty otherwise
         */
        public OptionalLong lastFetchTimestamp() {
            return lastFetchTimestamp;
        }

        /**
         * Return the lastCaughtUpTime in milliseconds for this replica.
         * @return The value of the lastCaughtUpTime if known, empty otherwise
         */
        public OptionalLong lastCaughtUpTimestamp() {
            return lastCaughtUpTimestamp;
        }

        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
            if (o == null || getClass() != o.getClass()) return false;
            ReplicaState that = (ReplicaState) o;
            return replicaId == that.replicaId
                && logEndOffset == that.logEndOffset
                && lastFetchTimestamp.equals(that.lastFetchTimestamp)
                && lastCaughtUpTimestamp.equals(that.lastCaughtUpTimestamp);
        }

        @Override
        public int hashCode() {
            return Objects.hash(replicaId, logEndOffset, lastFetchTimestamp, lastCaughtUpTimestamp);
        }

        @Override
        public String toString() {
            return "ReplicaState(" +
                "replicaId=" + replicaId +
                ", logEndOffset=" + logEndOffset +
                ", lastFetchTimestamp=" + lastFetchTimestamp +
                ", lastCaughtUpTimestamp=" + lastCaughtUpTimestamp +
                ')';
        }
    }
}
```

```
public class DescribeMetadataQuorumOptions extends AbstractOptions<DescribeMetadataQuorumOptions> {

}
```

DescribeMetadataQuorum Handler in the Admin Client

```
  /**
     * Describes the state of the metadata quorum.
     * <p>
     * This is a convenience method for {@link #describeMetadataQuorum(DescribeMetadataQuorumOptions)} with
default options.
     * See the overload for more details.
     *
     * @return the {@link DescribeMetadataQuorumResult} containing the result
     */
    default DescribeMetadataQuorumResult describeMetadataQuorum() {
        return describeMetadataQuorum(new DescribeMetadataQuorumOptions());
    }

    /**
     * Describes the state of the metadata quorum.
     * <p>
     * The following exceptions can be anticipated when calling {@code get()} on the futures obtained from
     * the returned {@code DescribeMetadataQuorumResult}:
     * <ul>
     *   <li>{@link org.apache.kafka.common.errors.ClusterAuthorizationException}
     *   If the authenticated user didn't have {@code DESCRIBE} access to the cluster.</li>
     *   <li>{@link org.apache.kafka.common.errors.TimeoutException}
     *   If the request timed out before the controller could list the cluster links.</li>
     * </ul>
     *
     * @param options The {@link DescribeMetadataQuorumOptions} to use when describing the quorum.
     * @return the {@link DescribeMetadataQuorumResult} containing the result
     */
    DescribeMetadataQuorumResult describeMetadataQuorum(DescribeMetadataQuorumOptions options);
```

Proposed change in the `DescribeQuorum` Response:

```
  "apiKey": 55,
  "type": "response",
  "name": "DescribeQuorumResponse",
  "validVersions": "0-1",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "ErrorCode", "type": "int16", "versions": "0+",
      "about": "The top level error code."},
    { "name": "Topics", "type": "[]TopicData",
      "versions": "0+", "fields": [
      { "name": "TopicName", "type": "string", "versions": "0+", "entityType": "topicName",
        "about": "The topic name." },
      { "name": "Partitions", "type": "[]PartitionData",
        "versions": "0+", "fields": [
        { "name": "PartitionIndex", "type": "int32", "versions": "0+",
          "about": "The partition index." },
        { "name": "ErrorCode", "type": "int16", "versions": "0+"},
        { "name": "LeaderId", "type": "int32", "versions": "0+", "entityType": "brokerId",
          "about": "The ID of the current leader or -1 if the leader is unknown."},
        { "name": "LeaderEpoch", "type": "int32", "versions": "0+",
          "about": "The latest known leader epoch"},
        { "name": "HighWatermark", "type": "int64", "versions": "0+"},
        { "name": "CurrentVoters", "type": "[]ReplicaState", "versions": "0+" },
        { "name": "Observers", "type": "[]ReplicaState", "versions": "0+" }
      ]}
    ]}],
  "commonStructs": [
      { "name": "ReplicaState", "versions": "0+", "fields": [
      { "name": "ReplicaId", "type": "int32", "versions": "0+", "entityType": "brokerId" },
      { "name": "LogEndOffset", "type": "int64", "versions": "0+",
        "about": "The last known log end offset of the follower or -1 if it is unknown"},
      { "name": "LastFetchTimestamp", "type": "int64", "versions": "1+", "ignorable": true, "default": -1,
        "about": "The last known leader wall clock time time when a follower fetched from the leader. This is
reported as -1 both for the current leader or if it is unknown for a voter"},
      { "name": "LastCaughtUpTimestamp", "type": "int64", "versions": "1+", "ignorable": true, "default": -1,
        "about": "The leader wall clock append time of the offset for which the follower made the most recent
fetch request. This is reported as the current time for the leader and -1 if unknown for a voter"}
    ]}
  ]
```

kafka-metadata-quorum.sh

The output from this API will also be captured in the `kafka-metadata-quorum.sh`. The output of the `--describe replication` command in the tool as defined in KIP-595 will change as follows:

```
> bin/kafka-metadata-quorum.sh --describe replication
ReplicaId   LogEndOffset    Lag               LastFetchTimestamp     LastCaughtUpTimestamp    Status
0           234134          0              tnow                           tnow                   Leader
1           234130          4              t2
t6                          Follower
2           234100          34             t3
t7                              Follower
3           234124          10             t4                                    t8
Observer
4           234130          4              t5
t9                          Observer
```

Proposed Changes

This KIP proposes exposing the `DescribeQuorum` API to the admin client and adding two new fields per replica **(including voters and observers)** to the `DescribeQuorum` API response.

These fields are intended to approximate the "time-lag" between the leader and the followers in the quorum.

1. **LastFetchTimestamp**
   This metric will be reported for each voter. This is a good approximation of the “liveness” of the voters and can be used to detect a network partition in the quorum.
   This information is already known to the leader for all voters and only needs to be added to the response
2. **LastCaughtUpTimestamp**
   This metric will be reported for each voter. This is akin to the metric used to track lag for replicas in ISR and it measures the approximate lag between the leader and the replica based on the offsets requested in the fetch requests and when they were made. The metric gives a measure of when was the last time a replica caught up to the leader. .
   To compute this metric, the Replica state maintains a few bits of information about fetch requests as they are received. Some of this information is not tracked by the Raft layer but it can be easily added in. The cost to track this information is minimal and it only requires some additional bookkeeping during pre-existing processing for a fetch.

   NOTE: Given the leader is always caught up to itself, the Last Caught Up Time for the leader will be the leader's wall clock time when it responded to the `DescribeQuorum` request.

Compatibility, Deprecation, and Migration Plan

The API response is versioned and the newly added fields will bump the message version on the response. This should handle any issues around compatibility.

Rejected Alternatives

Use Existing Fields

An alternative here was to not add in this information and use existing information available in the response to compute voter lag. As discussed in the motivation section, the only information available in the response at the moment is the offset, which does not allow for a very reliable mechanism to ascertain voter lag.

Track Some Other Information

Another possibility here is to add information which allows us to tell the voter lag more accurately. An argument can be made against the accuracy of the lag as measured by the additional fields proposed in this KIP. A more complete view of the voters would be to add in Replica Time/Offset Information