KIP-455: Create an Administrative API for Replica Reassignment

Master KIP

[KIP-500: Replace ZooKeeper with a Self-Managed Metadata Quorum](#) (Accepted)

Status

Current state: accepted

Discussion thread: here

JIRA: ⬆ **KAFKA-8345** - Create an Administrative API for Replica Reassignment `RESOLVED`

Release: controller-side changes in 2.4, command line changes in 2.6

Motivation

Currently, users initiate replica reassignment by writing directly to a ZooKeeper node named */admin/reassign_partitions*.

As explained in KIP-4, ZooKeeper based APIs have many problems. For example, there is no way to return a helpful error code if an invalid partition is proposed. Adding new features over time is difficult when external tools can access and overwrite Kafka's internal metadata data structures. Also, ZooKeeper-based APIs inherently lack security and auditability.

In addition to all the general problems of ZK-based APIs, the current reassignment interface has some problems specific to its particular structure. There is no mechanism provided for aborting a reassignment operation once it has been initiated. Furthermore, the API assumes that reassignment operations are launched as a batch – there is no way to incrementally add a new reassignment operation once the batch has been initiated.

We would like to provide a well-supported AdminClient API that does not suffer from these problems. Namely, it should support incremental replica reassignments and cancellation (revert) of ongoing reassignments.
This API can be used as a foundation on which to build future improvements.

Public Interfaces

AdminClient APIs

We will add two new admin APIs: alterPartitionAssignments, and listPartitionReassignments. As the names imply, the alter API modifies partition reassignments, and the list API lists the ones which are ongoing.

Unlike the current ZooKeeper-based API, alterPartitionAssignments can add or remove partition reassignments without interrupting unrelated assignments that are in progress. Partition reassignments can be added or removed by the API on a per-partition basis -- there are no global "before" or "after" snapshots.

```
/**
 * Change the reassignments for one or more partitions.
 * Providing an empty Optional (e.g via {@link Optional#empty()}) will <bold>cancel</bold> the reassignment for
the associated partition.
 *
 * @param reassignments    The reassignments to add, modify, or remove.
 * @param options          The options to use.
 * @return                 The result.
 */
public AlterPartitionReassignmentsResult alterPartitionReassignments(
        Map<TopicPartition, Optional<NewPartitionReassignment>> reassignments,
        AlterPartitionReassignmentsOptions options);

/**
 * A new partition reassignment, which can be applied via {@link AdminClient#alterPartitionReassignments(Map)}.
 */
public class NewPartitionReassignment {
    private final List<Integer> targetReplicas;
```

```
    /**
     * @throws IllegalArgumentException if no replicas are supplied
         */
    public NewPartitionReassignment(List<Integer> targetReplicas)

    public List<Integer> targetReplicas()
}

class AlterPartitionAssignmentsResult {
  Map<TopicPartition, KafkaFuture<Void>> values()
  Future<Void> all(); // Throws an exception if any reassignment was rejected
}

class AlterPartitionAssignmentsOptions extends AbstractOptions<> {
  // contains timeoutMs
}

/**
 * List some of the current partition reassignments.
 *
 * @param partitions     The partitions to show reassignments for. Must be non-null.
 * @param options        The options to use.
 */
public ListPartitionReassignmentsResult listPartitionReassignments(
      Set<TopicPartition> partitions,
      ListPartitionReassignmentsOptions options);

/**
 * List all of the current partition reassignments.
 *
 * @param options        The options to use.
 */
public ListPartitionReassignmentsResult listPartitionReassignments(
      ListPartitionReassignmentsOptions options);

abstract ListPartitionReassignmentsResult listPartitionReassignments(Optional<Set<TopicPartition>> partitions,
                                                    ListPartitionReassignmentsOptions options)

class ListPartitionReassignmentsOptions extends AbstractOptions<> {
  // contains timeoutMs
}

class ListPartitionReassignmentsResult {
  private final KafkaFuture<Map<TopicPartition, PartitionReassignment>> reassignments;
}

/**
 * A partition reassignment, which has been listed via {@link AdminClient#listPartitionReassignments()}.
 */
public class PartitionReassignment {
    /**
     * The brokers which this partition currently resides on.
     */
    private final List<Integer> replicas;

    /**
     * The brokers that we are adding this partition to as part of a reassignment.
     */
    private final List<Integer> addingReplicas;

    /**
     * The brokers that we are removing this partition from as part of a reassignment.
     */
    private final List<Integer> removingReplicas;
}
```

New RPCs

AlterPartitionReassignmentsRequest

This is used to create or cancel a set of partition reassignments. It must be sent to the controller.

This operation requires ALTER on CLUSTER.

Note that even though we have TimeoutMs, this field will be ignored in the implementation for the time being. Its implementation is tracked in

⬆ **KAFKA-8873** - Implement timeout for Alter/List PartitionReassignment APIs `OPEN`

```
{
  "apiKey": 45,
  "type": "request",
  "name": "AlterPartitionReassignmentsRequest",
  "validVersions": "0",
  "fields": [
    { "name": "TimeoutMs", "type": "int32", "versions": "0+", "default": "60000",
      "about": "The time in ms to wait for the request to complete." },
    { "name": "Topics", "type": "[]ReassignableTopic", "versions": "0+",
      "about": "The topics to reassign.", "fields": [
      { "name": "Name", "type": "string", "versions": "0+",
        "about": "The topic name." },
      { "name": "Partitions", "type": "[]ReassignablePartition", "versions": "0+",
        "about": "The partitions to reassign.", "fields": [
        { "name": "PartitionIndex", "type": "int32", "versions": "0+",
          "about": "The partition index." },
        { "name": "Replicas", "type": "[]int32", "versions": "0+", "nullableVersions": "0+", "default": "null",
          "about": "The replicas to place the partitions on, or null to cancel a pending reassignment for this
partition." }
      ]}
    ]}
  ]
}
```

AlterPartitionReassignmentsResponse

This is the response from AlterPartitionsReassignmentsRequest.

Possible top-level errors:

- REQUEST_TIMED_OUT: if the request timed out.
- NOT_CONTROLLER: if the node we sent the request to was not the controller.
- CLUSTER_AUTHORIZATION_FAILED: If we didn't have sufficient permission to perform the alteration.

Possible partition-level errors:

- INVALID_REPLICA_ASSIGNMENT - when the replica assignment is not valid (brokers are not part of the cluster, negative replica ids, empty assignment)
- NO_REASSIGNMENT_IN_PROGRESS - when a cancellation was called on a topic/partition which was not in the middle of a reassignment
- UNKNOWN_TOPIC_OR_PARTITION - when a topic/partition doesn't exist or a topic deletion is in progress

```
{
  "apiKey": 45,
  "type": "response",
  "name": "AlterPartitionReassignmentsResponse",
  "validVersions": "0",
  "fields": [
    { "name": "ThrottleTimeMs", "type": "int32", "versions": "0+",
      "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+",
      "about": "The error code." },
    { "name": "ErrorMessage", "type": "string", "versions": "0+", "nullableVersions": "0+",
      "about": "The error message, or null if there was no error." }
    { "name": "Responses", "type": "[]ReassignableTopicResponse", "versions": "0+",
      "about": "The responses to topics to reassign.", "fields": [
      { "name": "Name", "type": "string", "versions": "0+", "entityType": "topicName",
        "about": "The topic name" },
      { "name": "Partitions", "type": "[]ReassignablePartitionResponse", "versions": "0+",
        "about": "The responses to partitions to reassign", "fields": [
          { "name": "PartitionIndex", "type": "int32", "versions": "0+",
            "about": "The partition index." },
          { "name": "ErrorCode", "type": "int16", "versions": "0+",
            "about": "The error code." },
          { "name": "ErrorMessage", "type": "string", "versions": "0+", "nullableVersions": "0+",
            "about": "The error message, or null if there was no error." }
        ]
      ]
    }
  ]
}
```

ListPartitionReassignmentsRequest

This RPC lists the currently active partition reassignments. It must be sent to the controller.

It requires DESCRIBE on CLUSTER.

Note that even though we have TimeoutMs, this field will be ignored in the implementation for the time being. Its implementation is tracked in

⬆ **KAFKA-8873** - Implement timeout for Alter/List PartitionReassignment APIs `OPEN`

```
{
  "apiKey": 46,
  "type": "request",
  "name": "ListPartitionReassignmentsRequest",
  "validVersions": "0",
  "fields": [
    { "name": "TimeoutMs", "type": "int32", "versions": "0+", "default": "60000",
      "about": "The time in ms to wait for the request to complete." }
    { "name": "Topics", "type": "[]ListPartitionReassignmentsTopics", "versions": "0+", "nullableVersions":
"0+",
      "about": "The topics to list partition reassignments for, or null to list everything.", "fields": [
      { "name": "Name", "type": "string", "versions": "0+", "entityType": "topicName",
        "about": "The topic name" },
      { "name": "PartitionIndexes", "type": "[]int32", "versions": "0+",
        "about": "The partitions to list partition reassignments for" }
    ]}
  ]
}
```

ListPartitionReassignmentsResponse

Possible errors:

- REQUEST_TIMED_OUT: if the request timed out.
- NOT_CONTROLLER: if the node we sent the request to is not the controller.
- CLUSTER_AUTHORIZATION_FAILED: if we didn't have sufficient permissions.

If the top-level error code is set, no responses will be provided.

If a partition is not going through a reassignment, its AddingReplicas and RemovingReplicas fields will simply be empty.

If a partition doesn't exist, no response will be returned for it.

```
{
  "apiKey": 46,
  "type": "response",
  "name": "ListPartitionReassignmentsResponse",
  "validVersions": "0",
  "fields": [
    { "name": "ThrottleTimeMs", "type": "int32", "versions": "0+",
      "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+",
      "about": "The top-level error code, or 0 on success." },
    { "name": "ErrorMessage", "type": "string", "versions": "0+", "nullableVersions": "0+",
      "about": "The top-level error message, or null on success." },
    { "name": "Topics", "type": "[]OngoingTopicReassignment", "versions": "0+",
      "about": "The ongoing reassignments for each topic.", "fields": [
      { "name": "Name", "type": "string", "versions": "0+",
        "about": "The topic name." },
      { "name": "Partitions", "type": "[]OngoingPartitionReassignment", "versions": "0+",
        "about": "The ongoing reassignments for each partition.", "fields": [
        { "name": "PartitionIndex", "type": "int32", "versions": "0+",
          "about": "The index of the partition." },
        { "name": "Replicas", "type": "[]int32", "versions": "0+",
          "about": "The current replica set." },
        { "name": "AddingReplicas", "type": "[]int32", "versions": "0+",
          "about": "The set of replicas we are currently adding." },
        { "name": "RemovingReplicas", "type": "[]int32", "versions": "0+",
          "about": "The set of replicas we are currently removing." }
      ]}
    ]}
  ]
}
```

Modified RPCs

LeaderAndIsrRequest

We will add two new fields to LeaderAndIsrRequest named AddingReplicas and RemovingReplicas. These fields will, respectively, contain the replicas which are being added and removed from a partition's assignment.

These fields allow the controller to propagate reassignment information to the leader/follower brokers, paving the way for some of the improvements outlined in the Future Work section of this KIP. We will not make use of these fields as of this KIP.

```
diff --git a/clients/src/main/resources/common/message/LeaderAndIsrRequest.json b/clients/src/main/resources
/common/message/LeaderAndIsrRequest.json
index b8988351c..d45727078 100644
--- a/clients/src/main/resources/common/message/LeaderAndIsrRequest.json
+++ b/clients/src/main/resources/common/message/LeaderAndIsrRequest.json
@@ -20,6 +20,8 @@
   // Version 1 adds IsNew.
   //
   // Version 2 adds broker epoch and reorganizes the partitions by topic.
+  //
+  // Version 3 adds AddingReplicas and RemovingReplicas.
   "validVersions": "0-3",
   "fields": [
     { "name": "ControllerId", "type": "int32", "versions": "0+",
@@ -48,6 +50,8 @@
           "about": "The ZooKeeper version." },
         { "name": "Replicas", "type": "[]int32", "versions": "0+",
           "about": "The replica IDs." },
+        { "name": "AddingReplicas", "type": "[]int32", "versions": "3+",
+          "about": "The replica IDs that we are adding this partition to." },
+        { "name": "RemovingReplicas", "type": "[]int32", "versions": "3+",
+          "about": "The replica IDs that we are removing this partition from." },
         { "name": "IsNew", "type": "bool", "versions": "1+", "default": "false", "ignorable": true,
           "about": "Whether the replica should have existed on the broker or not." }
     ]}
```

Other Interfaces

NoReassignmentInProgressException

This exception is thrown when a user tries to cancel a reassignment which doesn't exist

```
package org.apache.kafka.common.errors;

/**
 * Thrown if a request cannot cancel a reassignment because one is not in progress.
 */
public class NoReassignmentInProgressException extends ApiException {
    public NoReassignmentInProgressException(String msg) {
        super(msg);
    }

    public NoReassignmentInProgressException(String msg, Throwable cause) {
        super(msg, cause);
    }
}
```

Proposed Changes

Cancellation

We propose supporting the "cancellation" of an on-going partition rebalance. It is essentially a rollback of a reassignment.
To concretize the notion - a cancellation of a reassignment means stopping any *on-going* partition movements and restoring the replica set to the one it was prior to the reassignment being initiated.
There is no guarantee that the reverted replica set will be in the same order as it was previously.

kafka-reassign-partitions.sh

kafka-reassign-partitions.sh is a script to create, inspect, or modify partition reassignments.

Deprecation of --zookeeper

We will deprecate the *--zookeeper* option in this script. New operations introduced by this KIP (cancellation, list) will not be supported by the ZK flag.

All old operations and new will be supported through the admin client. This will be initialized by the *--bootstrap-server* flag, which already exists.

The --list flag

There will be a new flag, *--list*, which will list all of the current replica reassignments that are going on in the cluster. For each reassignment, we will list the replica set, the adding replicas, and the removing replicas. If there are no ongoing replica reassignments, "No partition reassignments found." will be printed.

The --additional flag

Like the underlying KIP-455 API, the reassign-partitions tool will now work in an incremental fashion. Any new reassignment will be added on top of the current ones.

However, to avoid confusing users, the tool will decline to create a new reassignment if there is already one in progress. Users can choose to create additional reassignments even if there is one in progress by specifying the new --additional flag. This flag is only supported when --bootstrap-server is provided.

There is also one slight behavior change to --execute that is worth noting here. Previously, when --execute declined to create a new reassignment because one was already in progress, it would still apply the provided throttle, if any. In a world of concurrent reassignments, this behavior no longer makes sense, and has been removed. (However, users can still modify the throttle by simply supplying the --additional flag)

Cancellation with the tool

Additionally, we will support a *--cancel* flag. This flag will create a JSON file that, if applied, will cancel all on-going partition reassignments.

Not that there is a slight race condition here when using the tool. Someone may create a new reassignment in between creating the cancellation plan and applying it. This new reassignment would not be cancelled by the original cancellation plan. In general, we expect system administrators to stop any process or person that is creating new reassignments before cancelling all pending reassignments, so we don't expect this to be a problem in practice. If such processes are not stopped, the effect of cancelling all pending reassignments will be negated anyway, by the creation of new reassignments.

kafka-topics.sh

When describing partitions, kafka-topics.sh will show two additional fields, "addingReplicas," and "removingReplicas." These will have information about the adding and removing replicas for the given partition. *kafka-topics.sh* will get this information from *listReassigningPartitions.*

Implementation

ZooKeeper Changes

/brokers/topics/[topic]

We will store the reassignment data for each partition in this znode, as that is where we already store the current replicaSet for every partition.
This node will store two new maps, named *addingReplicas* and *removingReplicas*. Each key is the partition and the value is an array of integers denoting the replicas we should add or remove for that partition.

If both maps are empty, there is no on-going reassignment for that topic.

---

topics/[topic] zNode content

```
{
  "version": 2,
  "partitions": {"0": [1, 4, 2, 3] },
  "addingReplicas": {"0": [4] } # <----- NEW
  "removingReplicas": {"0": [1] }, # <---- NEW
}
```

The znode will now be at version 2 rather than 1. This change is backwards-compatible-- older brokers can read the new ZNode.

Controller Changes

When the controller starts up, it will load all of the currently active reassignments from the `*topics/[topic]*` znodes and the `*admin/reassign_partitions*` znode, with the latter taking precedence. This will not impose any additional startup overhead, because the controller already reads these znodes during start up.

The reason for having the `reassign_partitions` take precedence is that we cannot know for sure whether the Controller has taken action on that znode, so we allow it to overwrite the API-assigned reassignments in the event of a failover.

The controller will handle ListPartitionReassignments by listing the current active reassignments. It can supply this information directly from its in-memory cache. Note that this in-memory cache of all ongoing reassignments already existed prior to this KIP.

The controller will handle the AlterPartitionAssignmentsResponse RPC by modifying the /brokers/topics/[topic] znode. Once the node is modified, the controller will send out LeaderAndIsrRequest to start all the movements.

Algorithm

We will always add all the replicas in the "addingReplicas" field before starting to act on the "removingReplicas" field.

```
# Illustrating the rebalance of a single partition.
# R is the current replicas, I is the ISR list, AR is addingReplicas and RR is removingReplicas
R: [1, 2, 3], I: [1, 2, 3], AR: [], RR: []
# Reassignment called via API with targetReplicas=[4,3,2]

R: [1, 4, 3, 2], I: [1, 2, 3], AR: [4], RR: [1] # Controller sends LeaderAndIsr requests
# (We ignore RR until AR is empty)
R: [1, 4, 3, 2], I: [1, 2, 3, 4], AR: [4], RR: [1] # Leader 1 adds 4 to ISR
# The controller realizes that all the replicas in AR have been added and starts work on RR. Removes all of RR
from R and from the ISR, and sends StopReplica/LeaderAndIsr requests
R: [4, 3, 2], I: [2, 3, 4], AR: [], RR: [] # at this point the reassignment is considered complete
```

When all of the replicas in **AR** have been added, we remove **RR** from **R** and empty out **AR**/**RR** in one step.

If a new reassignment is issued during an on-going one, we cancel the current one by emptying out both **AR** and **RR**, send StopReplica requests to the replicas that were part of the old reassignment but not part of the new one, construct **AR**/**RR** from (the updated from the last-reassignment) **R** and **TR,** and start anew.

In general this algorithm is consistent with the current Kafka behavior - other brokers still get the full replica set consisting of both the original replicas and the new ones.

Here is an example of a cancellation:

```
# Illustrating the rebalance of a single partition.
# R is the current replicas, I is the ISR list, RR is removingReplicas and AR is addingReplicas
R: [1, 2, 3], I: [1, 2, 3], AR: [], RR: []
# Reassignment called via API with targetReplicas=[3, 4, 5]
R: [1, 2, 3, 4, 5], I: [1, 2, 3], AR: [4, 5], RR: [1, 2]
R: [1, 2, 3, 4, 5], I: [1, 2, 3, 4], AR: [4, 5], RR: [1, 2]
# Cancellation called
R: [1, 2, 3], I: [1, 2, 3], AR: [], RR: []
```

Essentially, once a cancellation is called we subtract AR from R, empty out both AR and RR, and send StopReplica/LeaderAndIsr requests to cancel the replica movements that have not yet completed.

Tool Changes

The *kafka-reassign-partitions.sh* tool will use the new AdminClient APIs to submit the reassignment plan. It will not use the old ZooKeeper APIs any more. In order to contact the admin APIs, the tool will accept a *--bootstrap-server* argument.

When changing the throttle, we will use *IncrementalAlterConfigs* rather than directly writing to Zookeeper.

Compatibility, Deprecation, and Migration Plan

/admin/reassign_partitions znode

For compatibility purposes, we will continue to allow assignments to be submitted through the *admin/reassign_partitions* node. Just as with the current code, this will only be possible if there are no current assignments. In other words, the znode has two states: empty and waiting for a write, and non-empty because there are assignments in progress. Once the znode is non-empty, further writes to it will be ignored.

Applications using the old API will not be able to cancel in-progress assignments. They will also not be able to monitor the status of in-progress assignments, except by polling the znode to see when it becomes empty, which indicates that no assignments are ongoing. To get these benefits, applications will have to be upgraded to the AdminClient API.

The deprecated ZooKeeper-based API will be removed in a future release.

There is one known edge-case with using both APIs which we will intentionally leave unaddressed:

```
1. znode reassignment involving partitionA and partitionB starts
2. API reassignment involving partitionA starts
3. Controller failover. It will effectively ignore the API reassignment in step 2. and move partitionA to the replica given
reassignment by the znode
```

There are ways to handle this edge-case, but we propose ignoring it due to the fact that:

- it is very unlikely for users to use both APIs at once
- we will be deprecating the znode reassignment API anyway
- it is not worth the complexity and performance hit from addressing such a rare and seemingly non-critical edge-case

Reassignments while upgrading/downgrading versions

In both scenarios, reassignments should continue to work because we still update and send the newly-updated, full replicaSet.

Rejected Alternatives

Store the pending replicas in a single Znode rather than in a per-partition ZNode

We could store the pending replicas in a single Znode per cluster, rather than putting them beside the replicas in /brokers/topics/[topic]. However, this Znode might become very large, which could lead to us running into maximum znode size problems. This is an existing scalability limitation which we would like to solve.

Another reason for storing the reassigning replicas state in the /brokers/topics/[topic] Znode is that this state belongs there, next to the replicaSet. By making it clear which replicas are pending and which are existing, we pave the way for future improvements in reassignment. For example, we will be able to distinguish between reassignment traffic and non-reassignment traffic for quota purposes, and provide better metrics.

Future Work

Reassignment Quotas that only throttle reassignment traffic

Currently, the quotas used for reassignment also apply to all non-ISR traffic. This is undesirable and can lead to problems if nodes fall out of the ISR.

Since this KIP makes brokers aware of which replicas are being used for reassignment, it will be possible to implement quotas that only hit reassignment traffic, and not all non-ISR traffic. This will also obviate the need to manually specify the list of reassigning partitions when setting the per-broker quotas.

Add reassignment metrics

We would like to provide metrics for reassignment. For example, we would like to know how many partitions are being reassigned on a given broker, what write and read bandwidth is being used, etc. Since this KIP clearly separates replicas that are being reassigned from those that are not, and makes this information available on each broker through the *LeaderAndIsrRequest*, this will be possible in the future.

Improved API for Reassignment Quotas

We would like a better API for managing reassignment quotas-- perhaps a new AdminClient function and RPC. This is somewhat of a minor improvement, since we already have a way to change the configuration via admin client. However, it would still be helpful.

Internal Batching

It would be nice if the controller could internally batch reassignment operations. The controller has more information with which to make decisions than external systems. If this were implemented, we would want to enrich the List API with more information, such as which reassignments were currently being worked on, and which the controller had decided to defer until later.

This could be added in a compatible way by making the default batch size infinite, so that users that didn't enable controller-side batching would get the current behavior of all reassignments starting immediately.