# KIP-860: Add client-provided option to guard against replication factor change during partition reassignments

## Status

**Current state**: **ACCEPTED**

**Discussion thread**: here
**Vote thread**: here

**JIRA**: KAFKA-14121

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

## Motivation

Many times, we don't have the intention of changing the replication factor of a partition during a reassignment.

Using Kafka's public APIs to get metadata regarding the non-reassigning replicas for a topic is unreliable and prone to race conditions.
If a person or a system is to rely on the provided metadata, it can end up unintentionally increasing the replication factor for a partition.
It would be useful to have some sort of guardrail against this happening inadvertently.

### Race Condition Explained

AdminClient#describeTopics returns the full replica set for any particular partitions, without distinguinshing whether the partition is in the process of being reassigned or not. The result is that the TopicPartitionInfo#replicas field contains the full intermediate replica set during a reassignment, and the caller isn't able to accurately determine at that same point of time that a reassignment is happening.

AdminClient#listPartitionReassignments returns the set of partitions that are being reassigned. In theory, a caller could use this API to understand what partitions are reassigning and only later describe them once they aren't reassigning. Alternatively, they could use the reassignment information from the response to understand what the replicas of the underlying partition are/will be.

Our kafka-topics.sh tool, for example, calls both APIs to show us information regarding the topic. It 1) describes the topic and 2) lists the reassignments, and then shows the detailed replica set to the user - ("replicas: `[1,2,3,4,5], addingReplicas: [4,5], removingReplicas: [1,2])`

A fundamental race condition exists, though, where one (person, automated system, kafka-tools CLI) could inaccurately describe a topic believing that there are no reassignments happening and believe the intermediate reassignment replica-set to be the finalized replica set.
Because listPartitionReassignments goes to the controller, it is always guaranteed to have the latest metadata regarding reassignments. describeTopics, on the other hand, goes to any broker - and due to Kafka's async metadata propagation it is therefore vulnerable to returning stale metadata.
Race condition cases exist where listPartitionReassignments can say that there are no on-going reassignments, and describeTopics can return a replica set denoting the reassigning replicas.

Two examples of this:

1. partition `tp-0` has replicas `[1,2,3]` . AdminClient#alterPartitionReassignments is called to reassign to `[4,5,6]` .
2. `AdminClient#describeTopics("tp")` returns a replica set of `[1,2,3,4,5,6]`
3. The reassignment completes
4. `AdminClient#listPartitionReassignments()` returns an empty set - nothing is being reassigned

-

1. partition `tp-0` has replicas `[1,2,3]` . AdminClient#alterPartitionReassignments is called to reassign to `[4,5,6]` .
2. The reassignment completes
3. `AdminClient#listPartitionReassignments()` returns an empty set - nothing is being reassigned
4. `AdminClient#describeTopics("tp")` goes to a node with metadata that's not yet updated, returns a replica set of `[1,2,3,4,5,6]`

In both cases, the caller is left to assume that "tp" has a replication factor of 6 and a replica set of `[1,2,3,4,5,6]` when that is obviously not the case. If the caller then wants to reassign the partition, it may inaccurately assume an RF of 6 and in the hopes of preserving the RF, reassign the partition to a new set of 6 replicas, inadvertently solidifying the RF=6 in the process.

### Occurrences in the wild

In particular, we have consistently hit the aforementioned race conditions many times in internal Confluent software that is meant to automate continuous partition reassignment for a cluster. It leaves us unable to rely on the public APIs to reliably determine the right assignment for a partition.

While we lack direct experience to confirm this, we speculate that the same is possible in the popular open source reassignment software Cruise Control, where Metadata{Request,Response} are used to collect metadata for the cluster. Since said requests also return the intermittent replica set and do not contain explicit reassignment information, the same problem should be theoretically possible.
Issue 1879 in their GitHub seems to point to this very problem, and the fix they implemented was to check against inconsistency between the metadata and the cluster's metrics they collect.

We propose adding an option to fail-fast in the case where a replication factor is changed without the desire for it.

# Public Interfaces

## Admin API

The `AlterPartitionReassignmentsOptions` will be extended to allow the configuration of a new boolean:

**new AlterPartitionReassignmentsOptions field**

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

  private Boolean allowReplicationFactorChange = true;

  /**
   * Set the option indicating if the alter partition reassignments call should be
   * allowed to alter the replication factor of a partition.
   * In cases where it is not allowed, any replication factor change will result in an exception thrown by the
API.
   */
  public boolean allowReplicationFactorChange(boolean allow) {
    this.allowReplicationFactorChange = allow;
    return allowReplicationFactorChange;
  }

  /**
   * A boolean indicating if the alter partition reassignments should be
   * allowed to alter the replication factor of a partition.
   * In cases where it is not allowed, any replication factor change will result in an exception thrown by the
API.
   */
  public boolean allowReplicationFactorChange() {
    return this.allowReplicationFactorChange;
  }
}
```

The AlterPartitionReassignments Admin API will begin throwing the `InvalidReplicationFactorException` and `UnsupportedVersionException` . Its documentation will be updated to mention that:

**Admin#alterPartitionReassignments documentation**

```
diff --git a/clients/src/main/java/org/apache/kafka/clients/admin/Admin.java b/clients/src/main/java/org/apache
/kafka/clients/admin/Admin.java
index 911dd533a0..fb318fbca5 100644
--- a/clients/src/main/java/org/apache/kafka/clients/admin/Admin.java
+++ b/clients/src/main/java/org/apache/kafka/clients/admin/Admin.java
@@ -1085,6 +1085,9 @@ public interface Admin extends AutoCloseable {
      *   if the request timed out before the controller could record the new assignments.</li>
      *   <li>{@link org.apache.kafka.common.errors.InvalidReplicaAssignmentException}
      *   If the specified assignment was not valid.</li>
+     *   <li>{@link org.apache.kafka.common.errors.InvalidReplicationFactorException}
+     *   If the replication factor was changed in an invalid way.
+     *   Only thrown when {@link AlterPartitionReassignmentsOptions#allowReplicationFactorChange()} is set to
false and the request is attempting to alter reassignments (not cancel)</li>
+     *   <li>{@link org.apache.kafka.common.errors.UnsupportedVersionException}
+     *   If {@link AlterPartitionReassignmentsOptions#allowReplicationFactorChange()} was changed outside the
default
+     *   and the server does not support the option (e.g due to an old Kafka version).</li>
      *   <li>{@link org.apache.kafka.common.errors.NoReassignmentInProgressException}
      *   If there was an attempt to cancel a reassignment for a partition which was not being reassigned.</li>
      * </ul>
```

Note that during uses of AlterPartitionReassignment for **cancelling** reassignments, the option will have no effect (effectively be ignored)

## CLI

Accordingly, the kafka-reassign-partitions.sh tool will be updated to allow supplying the new option:

**ReassignPartitionsCommand.scala**

```
    val disallowReplicationFactorChange = parser.accepts("disallow-replication-factor-change", "Denies the
ability to change a partition's replication factor as part of this reassignment through adding validation
against it.")
```

and a sample usage would be:

```
/bin/kafka-reassign-partitions.sh --reassignment-json-file ./reassign.json --execute --bootstrap-server
localhost:9092 --disallow-replication-factor-change
```

## Proposed Changes

### RPC

A new option will be added to the request object and its associated request/response version bumped to 1.

### AlterPartitionReassignmentsRequest

```
// Version 1 adds the ability to allow/disallow changing the replication factor as part of the request.
{
  "apiKey": 45,
  "type": "request",
  "listeners": ["broker", "controller", "zkBroker"],
  "name": "AlterPartitionReassignmentsRequest",
  "validVersions": "0-1", // <-------NEW - version 1
  "flexibleVersions": "0+",
  "fields": [
    { "name": "TimeoutMs", "type": "int32", "versions": "0+", "default": "60000",
      "about": "The time in ms to wait for the request to complete." },
    { "name": "AllowReplicationFactorChange", "type": "boolean", "versions": "1+", "default": "true", //
<--------- NEW
      "about": "The option indicating whether changing the replication factor of any given partition as part of
this request is a valid move." }, // <--------- NEW
    { "name": "Topics", "type": "[]ReassignableTopic", "versions": "0+",
      "about": "The topics to reassign.", "fields": [
      { "name": "Name", "type": "string", "versions": "0+", "entityType": "topicName",
        "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",
"entityType": "brokerId",
          "about": "The replicas to place the partitions on, or null to cancel a pending reassignment for this
partition." }
      ]}
    ]}
  ]
}
```

### AlterPartitionReassignmentsResponse

```
// Version 1 adds the ability to allow/disallow changing the replication factor as part of the request.
{
  "apiKey": 45,
  "type": "response",
  "name": "AlterPartitionReassignmentsResponse",
  "validVersions": "0-1", // <-------NEW - version 1
  "flexibleVersions": "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": "AllowReplicationFactorChange", "type": "boolean", "versions": "1+", "default": "true",
"ignorable": true, // <--------- NEW
      "about": "The option indicating whether changing the replication factor of any given partition as part of
the request was allowed." }, // <--------- NEW
    { "name": "ErrorCode", "type": "int16", "versions": "0+",
      "about": "The top-level error code, or 0 if there was no error." },
    { "name": "ErrorMessage", "type": "string", "versions": "0+", "nullableVersions": "0+",
      "about": "The top-level 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 for this partition, or 0 if there was no error." },
        { "name": "ErrorMessage", "type": "string", "versions": "0+", "nullableVersions": "0+",
          "about": "The error message for this partition, or null if there was no error." }
      ]}
    ]}
  ]
}
```

## Validation

If AllowReplicationFactorChange is false, and the request does change the replication factor of a partition, a `new ApiError(Errors.INVALID_REPLICATION_FACTOR)` will be returned for the associated partitions. Consistent with the current behavior of the API, valid (unaffected) partition reassignments will succeed as part of the same request - and only the invalid ones will have an error returned as part of the response.

The validation itself will be pretty simple.

- If the targetReplicaSet of the reassignment differs from the current replica set of the partition, an error is thrown.
- In the case of an already-reassigning partition being reassigned again, the validation compares the targetReplicaSet size of the reassignment to the targetReplicaSet size of the new reassignment and throws if those differ.

This validation will NOT apply when altering partitions through ZooKeeper. This KIP does not propose a way to pass that option when reassigning through direct ZK changes, mainly because that way altering is being phased out as part of KIP-500 (it is not available in kafka-reassign-partitions.sh since Kafka 3.0)

## Compatibility Validation

If a new client version that supports this option tries to set it on a Kafka broker with an old version that doesn't support this option, we want to throw an exception.
To enforce this, we will add a client-side validation of a minimum request version of `(1)` in the request builder in the admin client. This results in the client throwing an `UnsupportedVersionException("The broker does not support ALTER_PARTITION_REASSIGNMENTS with version in range [1, 1]. The supported range is [0, 0].")`. We will modify this exception to have explicit wording about the unsupported AllowReplicationFactorChange option in order to make it clear to the user what's causing the error (e.g "The broker does not support the AllowReplicationFactorChange option for the AlterPartitionReassignments API. Consider re-sending the request without the option or updating the server version")

This client-side validation will only be performed when the option (allowReplicationFactorChange) is set to a non-default value (false).

# Compatibility, Deprecation, and Migration Plan

### Client Backward Compatibility

The change should be backward compatible – i.e it should work if we have an old client version using a new Kafka version, because we keep the default option (true) consistent with the behavior today - allowing callers to change the replication factor of a partition without a problem.

### Client Forward Compatibility

If we have a new client version using an old Kafka version, the API should continue working while it's set to work with the default value. If a user sets the allowReplicationFactor to the non-default value of false, the client will throw an exception in order to inform the user that the operation will not be honored by the broker. Users are then free to handle this as they desire - either retry the operation without the flag, or abort.

# Test Plan

An integration test, alongside unit tests, will be added to verify the behavior works as expected.

# Rejected Alternatives

There are many ways to solve this problem, all with varying scope and effort required. Let us go over some of these in order of their complexity:

## Add server side option

Instead of the client voluntarily choosing to have this validation, the server can enforce it without consideration. A server-side config can be added, e.g `partition.reassignments.replication.factor.change.allowed={false,true}` acting in the same way as the client-side option described here, with the only difference being that the client has no way of circumventing it.

This can be useful in Kafka setups where an administrator wants to explicitly limit what the users can do, but one could also reason that a well-abstracted Kafka setup would not allow users to alter much of the cluster's configurations, including things like replica assignments.

We chose to go with the client-side option, as the motivation described in this KIP explains a way in which a client unintentionally changes the RF of a partition.

We could not think of situations where an administrator would want to allow a user to change a partition's assignments, but not want them to change its replication factor. Further, any such validations may have increased scope as they may need to be added to the CreatePartitions API

It is worth noting that both options aren't mutually exclusive. We can add the client-side option and, if deemed necessary, add a server-side override to this later.

## Compare-and-set approach: validate source replicas

An alternative way of achieving a stronger set of validation would be to draw inspiration from the `AtomicInteger#compareAndSet()` API.
We could overload the AdminClient API to allow passing in an expected assignment as part of each partition's reassignment

```
AlterPartitionReassignmentsResult alterPartitionReassignments(
        Map<TopicPartition,
                 Map<Optional<ExpectedPartitionAssignment>, Optional<NewPartitionReassignment>>
        > reassignments)
```

The server would then only reassign to the desired target replica set if the partition matches the expected assignment. In cases where it doesn't match, it would throw a new type of exception and refuse reassigning the partitio.
In cases where a reassignment was already in progress, the server could either compare the existing reassignment's target replica set to the expected one, or compare the existing intermediate reassignment replica set to the expected one.

This proposal offers a stronger guarantee beyond simple replication factor changes, as it allows a caller the guarantee that the reassignment he's doing will trigger a move exact to his expectations.

We were initially thinking of proposing this approach and are still on the fence with regards to which approach should be adopted.
We chose to reject this alternative mainly because it results in a more complex UX (especially in the reassign partitions CLI case) but are happy to discuss the tradeoffs with the community and make a call.

## Return Consistent Metadata

The root cause that motivated this KIP is the inconsistency in the way our APIs expose reassignment metadata. If the describeTopics API could return reassigning metadata (addingReplicas/removingReplicas), this exact problem of inadvertently increasing the replication factor would not exist.

We choose to refrain from proposing any such solution as the level of effort in designing and implementing such a change is much greater than adding a guardrail in the reassignment API. At a minimum, it would involve changing the UpdateMetadataRequest and whatever equivalent is used in KRaft to propagate topic metadata across all nodes, and have a greater scope of backward compatibility concerns/public API changes.