KIP-464: Defaults for AdminClient#createTopic

Status

Current state: *Adopted*

Discussion thread: here

JIRA: KAFKA-8305 - Getting issue details... STATUS *(2.3) and* KAFKA-8531 - Getting issue details... STATUS *(2.4)*

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

Motivation

Many simple workloads on Kafka can benefit from default partitioning and replication configurations. This allows system administrators to set some sane defaults, making Kafka more accessible to engineers in an organization that may not have knowledge to set meaningful values for these configurations.

There are two existing Kafka cluster configurations that we well leverage:

- `**num.partitions**` - the default number of log partitions per topic
- `**default.replication.factor**` - the default replication factor for a topic

Today, these two configurations can only be leveraged from auto.topic.create; this KIP proposes leveraging them from the AdminClient APIs as well.

Public Interfaces

We will add one new constructor to **NewTopic** that resolves all defaults and requires only the topic name:

```
public NewTopic(String name); // uses default partitions and replication factor from broker config
```

This (NewTopic) is what will be passed in **AdminClient#createTopics**. The default values will remain the same (sentinel value of -1).

The `CreateTopicsRequest` will bump its protocol version to 4 so that any requests sent to old brokers with the default values and no replica assignments will fail with a UnsupportedVersionException.

To exploit this new feature in KafkaStreams, we update the default value of Streams configuration parameter `replication.factor` from `1` to `-1`. Cf KIP-733: change Kafka Streams default replication factor config

Proposed Changes

The change is straightforward, **AdminManager** on the broker will modify its verification logic to the following:

- (no change) If partitions and replicas are both present (but manual assignment is not), we use the partitions and replicas and determine assignment automatically
- (no change) If partitions and replicas are both absent and a manual assignment is present, then we use the manual assignment
- (no change) If either partitions or replicas are present and a manual assignment is present, the request fails
- ~~(new) if only partitions is present, we will no longer fail and instead use the supplied partitions and the default replication factor~~
- ~~(new) if only replicas are present, we will no longer fail and instead use the supplied replication factor and the default partitions~~
- (new) if neither partitions, replicas or manual assignment are present, we will no longer fail and instead use the default replication factor and default partitions

The same error codes will be used to convey error:

```
    INVALID_PARTITIONS(37, "Number of partitions is below 1.",
            InvalidPartitionsException::new),
    INVALID_REPLICATION_FACTOR(38, "Replication factor is below 1 or larger than the number of available
brokers.",
            InvalidReplicationFactorException::new),
    INVALID_REPLICA_ASSIGNMENT(39, "Replica assignment is invalid.",
```

Compatibility, Deprecation, and Migration Plan

- No request that was previously valid will become invalid, and no request that was previously valid will change behavior.
- Some previously invalid requests (e.g. valid partitions and no replication factor or manual assignments) will now succeed.
- Sending a request with default partitions/replicas to an old broker will throw `UnsupportedVersionException`

Rejected Alternatives

- Add a new configuration to differentiate **default.replication.factor** for auto-topic-create and for normal behavior (i.e. feature flagging this change). I believe this will just confuse users, and the current configuration name is descriptive enough.
- Change the broker protocol to accept a new type of request that does not have any value (instead of a sentinel value) for the replication factor /partitions.
    - The original proposal in this KIP will be easier to perform upgrades - since all Kafka brokers will be able to deserialize the incoming request, the only difference is whether or not the request succeeds (i.e. creates the topic) or fails (i.e. sends an error message).

Below Options may be implemented in a follow-up KIP

Option 2: Partial Port of NewTopicBuilder

We can add the following builder to org.apache.kafka.clients.admin and have the existing one in the connect module extend from it to maintain the existing methods:

```
public static Builder build(String name) { return new Builder(name)); }

  public static class Builder {

     public Builder(String name); // this will be called from NewTopic#build(String)

    /**
     * Specify the desired number of partitions for the topic.
     *
     * @param numPartitions the desired number of partitions; must be positive
     * @return this builder to allow methods to be chained; never null
     */
     public Builder partitions(int numPartitions);

     /**
      * Specify the desired replication factor for the topic.
      *
      * @param replicationFactor the desired replication factor; must be positive
      * @return this builder to allow methods to be chained; never null
      */
     public Builder replicationFactor(short replicationFactor);

      /**
       * Specify the configuration properties for the topic, overwriting any previously-set properties.
       *
       * @param configs the desired topic configuration properties, or null if all existing properties should
be cleared
       * @return this builder to allow methods to be chained; never null
       */
     public Builder config(Map<String, Object> configs);

      /**
       * Build the {@link NewTopic} representation.
       *
       * @return the topic description; never null
       */
     public NewTopic build();
  }
}
```

Option 3: Complete Port of NewTopicBuilder

We will make the **NewTopicBuilder** a first class API for constructing topics with any permutation of defaults and specific configurations. This moves the following API from the org.apache.kafka**.connect.runtime** package to the org.apache.kafka**.clients.admin** module as an api within **NewTopic**, *leaving the old one in place but deprecating it for compatibility* (the usage within kafka connect will be migrated to use the one in the new package):

```
package org.apache.kafka.clients.admin;

...

public class NewTopic {

  ...

public static Builder build(String name) { return new Builder(name)); }

  public static class Builder {

     public Builder(String name); // this will be called from NewTopic#build(String)

    /**
     * Specify the desired number of partitions for the topic.
     *
     * @param numPartitions the desired number of partitions; must be positive
     * @return this builder to allow methods to be chained; never null
     */
```

```
     public Builder partitions(int numPartitions);

     /**
      * Specify the desired replication factor for the topic.
      *
      * @param replicationFactor the desired replication factor; must be positive
      * @return this builder to allow methods to be chained; never null
      */
     public Builder replicationFactor(short replicationFactor);

      /**
       * Specify that the topic should be compacted.
       *
       * @return this builder to allow methods to be chained; never null
       */
      public Builder compacted();

      /**
       * Specify the minimum number of in-sync replicas required for this topic.
       *
       * @param minInSyncReplicas the minimum number of in-sync replicas allowed for the topic; must be positive
       * @return this builder to allow methods to be chained; never null
       */
      public Builder minInSyncReplicas(short minInSyncReplicas);

      /**
       * Specify whether the broker is allowed to elect a leader that was not an in-sync replica when no ISRs
       * are available.
       *
       * @param allow true if unclean leaders can be elected, or false if they are not allowed
       * @return this builder to allow methods to be chained; never null
       */
      public Builder uncleanLeaderElection(boolean allow);

      /**
       * Specify the configuration properties for the topic, overwriting any previously-set properties.
       *
       * @param configs the desired topic configuration properties, or null if all existing properties should
be cleared
       * @return this builder to allow methods to be chained; never null
       */
     public Builder config(Map<String, Object> configs);

      /**
       * Build the {@link NewTopic} representation.
       *
       * @return the topic description; never null
       */
     public NewTopic build();
  }
}
```