KIP-476: Add Java AdminClient Interface

Status

Current state: *Adopted*

Discussion thread: *here*

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

The Kafka AdminClient is currently an abstract class. The original motivation for using an abstract class, rather than an interface, was because Java 7 did not support default method for interfaces. Kafka now has a minimum Java version of 1.8, which does support default methods on interface.

An AdminClient interface has several advantages over an abstract base class, most notably allowing multi-inheritance and the use of dynamic proxies. The current use of an abstract base class imposes unnecessary restrictions on client code making use of the AdminClient and other libraries which accept the AdminClient.

As an example, a project could make use of a dynamic proxy on an AdminClient interface to ensure downstream code only called read-only methods on the client.

Public Interfaces

The following changes to the public interfaces of the Kafka libraries will be made:

A new `Admin` interface will be added, (naming inline with the client interfaces `Consumer` and `Producer`). The interface will define all the methods the `AdminClient` class currently defines, with default implementations where necessary.

The existing `AdminClient` abstract class will be changed to implement the new `Admin` interface and its own implementation remove, with the exception of the factory methods, which will be maintained for backwards compatibility. The `Admin` interface will also have factory methods, but will return `Admin` instances, not `AdminClient` instances, so the factory methods are required on the `AdminClient` to avoid compilation errors for users.

All use of `AdminClient` within the Kafka project will be replaced with use of the `Admin` interface, with the exception of any classes extending the abstract class, (`KafkaAdminClient` and `MockAdminClient`), which will continue to do so to maintain backwards compatibility, and Kafka Stream's `KafkaClientSupplier`, which has a `getAdminClient` method that must continue to return `AdminClient` to maintain binary compatibility. `KafkaClientSupplier` will have a new `Admin getAdmin` method added and the old one deprecated and its internal use switched to the new method.

AdminClient.java

```
@InterfaceStability.Evolving
public abstract class AdminClient implements Admin {

    /**
     * Create a new AdminClient with the given configuration.
     *
     * @param props The configuration.
     * @return The new KafkaAdminClient.
     */
    public static AdminClient create(Properties props) {
        return KafkaAdminClient.createInternal(new AdminClientConfig(props, true), null);
    }

    /**
     * Create a new AdminClient with the given configuration.
     *
     * @param conf The configuration.
     * @return The new KafkaAdminClient.
     */
    public static AdminClient create(Map<String, Object> conf) {
        return KafkaAdminClient.createInternal(new AdminClientConfig(conf, true), null);
    }
}
```

Admin.java

```
@InterfaceStability.Evolving
public interface Admin extends AutoCloseable {

    /**
     * Create a new AdminClient with the given configuration.
     *
     * @param props The configuration.
     * @return The new KafkaAdminClient.
     */
    static Admin create(Properties props) {
        return KafkaAdminClient.createInternal(new AdminClientConfig(props, true), null);
    }

    /**
     * Create a new AdminClient with the given configuration.
     *
     * @param conf The configuration.
     * @return The new KafkaAdminClient.
     */
    static Admin create(Map<String, Object> conf) {
        return KafkaAdminClient.createInternal(new AdminClientConfig(conf, true), null);
    }

    /**
     * Close the AdminClient and release all associated resources.
     *
     * See {@link Admin#close(long, TimeUnit)}
     */
    @Override
    default void close() {
        close(Long.MAX_VALUE, TimeUnit.MILLISECONDS);
    }

    /**
     * Close the AdminClient and release all associated resources.
     *
     * The close operation has a grace period during which current operations will be allowed to
     * complete, specified by the given duration and time unit.
     * New operations will not be accepted during the grace period.  Once the grace period is over,
```

```
     * all operations that have not yet been completed will be aborted with a TimeoutException.
     *
     * @param duration  The duration to use for the wait time.
     * @param unit      The time unit to use for the wait time.
     * @deprecated Since 2.2. Use {@link #close(Duration)} or {@link #close()}.
     */
    @Deprecated
    default void close(long duration, TimeUnit unit) {
        close(Duration.ofMillis(unit.toMillis(duration)));
    }

    /**
     * Close the Admin client and release all associated resources.
     *
     * The close operation has a grace period during which current operations will be allowed to
     * complete, specified by the given duration.
     * New operations will not be accepted during the grace period.  Once the grace period is over,
     * all operations that have not yet been completed will be aborted with a TimeoutException.
     *
     * @param timeout  The time to use for the wait time.
     */
    void close(Duration timeout);

    /**
     * Create a batch of new topics with the default options.
     *
     * This is a convenience method for #{@link #createTopics(Collection, CreateTopicsOptions)} with default
options.
     * See the overload for more details.
     *
     * This operation is supported by brokers with version 0.10.1.0 or higher.
     *
     * @param newTopics         The new topics to create.
     * @return                  The CreateTopicsResult.
     */
    default CreateTopicsResult createTopics(Collection<NewTopic> newTopics) {
        return createTopics(newTopics, new CreateTopicsOptions());
    }

    /**
     * Create a batch of new topics.
     *
     * This operation is not transactional so it may succeed for some topics while fail for others.
     *
     * It may take several seconds after {@code CreateTopicsResult} returns
     * success for all the brokers to become aware that the topics have been created.
     * During this time, {@link Admin#listTopics()} and {@link Admin#describeTopics(Collection)}
     * may not return information about the new topics.
     *
     * This operation is supported by brokers with version 0.10.1.0 or higher. The validateOnly option is
supported
     * from version 0.10.2.0.
     *
     * @param newTopics         The new topics to create.
     * @param options           The options to use when creating the new topics.
     * @return                  The CreateTopicsResult.
     */
    CreateTopicsResult createTopics(Collection<NewTopic> newTopics, CreateTopicsOptions options);

    /**
     * This is a convenience method for #{@link Admin#deleteTopics(Collection, DeleteTopicsOptions)}
     * with default options. See the overload for more details.
     *
     * This operation is supported by brokers with version 0.10.1.0 or higher.
     *
     * @param topics            The topic names to delete.
     * @return                  The DeleteTopicsResult.
     */
    default DeleteTopicsResult deleteTopics(Collection<String> topics) {
        return deleteTopics(topics, new DeleteTopicsOptions());
    }
```

```
/**
 * Delete a batch of topics.
 *
 * This operation is not transactional so it may succeed for some topics while fail for others.
 *
 * It may take several seconds after the {@code DeleteTopicsResult} returns
 * success for all the brokers to become aware that the topics are gone.
 * During this time, AdminClient#listTopics and AdminClient#describeTopics
 * may continue to return information about the deleted topics.
 *
 * If delete.topic.enable is false on the brokers, deleteTopics will mark
 * the topics for deletion, but not actually delete them.  The futures will
 * return successfully in this case.
 *
 * This operation is supported by brokers with version 0.10.1.0 or higher.
 *
 * @param topics            The topic names to delete.
 * @param options           The options to use when deleting the topics.
 * @return                  The DeleteTopicsResult.
 */
DeleteTopicsResult deleteTopics(Collection<String> topics, DeleteTopicsOptions options);

/**
 * List the topics available in the cluster with the default options.
 *
 * This is a convenience method for #{@link Admin#listTopics(ListTopicsOptions)} with default options.
 * See the overload for more details.
 *
 * @return                  The ListTopicsResult.
 */
default ListTopicsResult listTopics() {
    return listTopics(new ListTopicsOptions());
}

/**
 * List the topics available in the cluster.
 *
 * @param options           The options to use when listing the topics.
 * @return                  The ListTopicsResult.
 */
ListTopicsResult listTopics(ListTopicsOptions options);

/**
 * Describe some topics in the cluster, with the default options.
 *
 * This is a convenience method for #{@link Admin#describeTopics(Collection, DescribeTopicsOptions)} with
 * default options. See the overload for more details.
 *
 * @param topicNames        The names of the topics to describe.
 *
 * @return                  The DescribeTopicsResult.
 */
default DescribeTopicsResult describeTopics(Collection<String> topicNames) {
    return describeTopics(topicNames, new DescribeTopicsOptions());
}

/**
 * Describe some topics in the cluster.
 *
 * @param topicNames        The names of the topics to describe.
 * @param options           The options to use when describing the topic.
 *
 * @return                  The DescribeTopicsResult.
 */
DescribeTopicsResult describeTopics(Collection<String> topicNames, DescribeTopicsOptions options);

/**
 * Get information about the nodes in the cluster, using the default options.
 *
 * This is a convenience method for #{@link Admin#describeCluster(DescribeClusterOptions)} with default
```

```
options.
     * See the overload for more details.
     *
     * @return                 The DescribeClusterResult.
     */
    default DescribeClusterResult describeCluster() {
        return describeCluster(new DescribeClusterOptions());
    }

    /**
     * Get information about the nodes in the cluster.
     *
     * @param options          The options to use when getting information about the cluster.
     * @return                 The DescribeClusterResult.
     */
    DescribeClusterResult describeCluster(DescribeClusterOptions options);

    /**
     * This is a convenience method for #{@link Admin#describeAcls(AclBindingFilter, DescribeAclsOptions)} with
     * default options. See the overload for more details.
     *
     * This operation is supported by brokers with version 0.11.0.0 or higher.
     *
     * @param filter           The filter to use.
     * @return                 The DeleteAclsResult.
     */
    default DescribeAclsResult describeAcls(AclBindingFilter filter) {
        return describeAcls(filter, new DescribeAclsOptions());
    }

    /**
     * Lists access control lists (ACLs) according to the supplied filter.
     *
     * Note: it may take some time for changes made by createAcls or deleteAcls to be reflected
     * in the output of describeAcls.
     *
     * This operation is supported by brokers with version 0.11.0.0 or higher.
     *
     * @param filter           The filter to use.
     * @param options          The options to use when listing the ACLs.
     * @return                 The DeleteAclsResult.
     */
    DescribeAclsResult describeAcls(AclBindingFilter filter, DescribeAclsOptions options);

    /**
     * This is a convenience method for #{@link Admin#createAcls(Collection, CreateAclsOptions)} with
     * default options. See the overload for more details.
     *
     * This operation is supported by brokers with version 0.11.0.0 or higher.
     *
     * @param acls             The ACLs to create
     * @return                 The CreateAclsResult.
     */
    default CreateAclsResult createAcls(Collection<AclBinding> acls) {
        return createAcls(acls, new CreateAclsOptions());
    }

    /**
     * Creates access control lists (ACLs) which are bound to specific resources.
     *
     * This operation is not transactional so it may succeed for some ACLs while fail for others.
     *
     * If you attempt to add an ACL that duplicates an existing ACL, no error will be raised, but
     * no changes will be made.
     *
     * This operation is supported by brokers with version 0.11.0.0 or higher.
     *
     * @param acls             The ACLs to create
     * @param options          The options to use when creating the ACLs.
     * @return                 The CreateAclsResult.
     */
```

```
    CreateAclsResult createAcls(Collection<AclBinding> acls, CreateAclsOptions options);

    /**
     * This is a convenience method for #{@link Admin#deleteAcls(Collection, DeleteAclsOptions)} with default
options.
     * See the overload for more details.
     *
     * This operation is supported by brokers with version 0.11.0.0 or higher.
     *
     * @param filters           The filters to use.
     * @return                  The DeleteAclsResult.
     */
    default DeleteAclsResult deleteAcls(Collection<AclBindingFilter> filters) {
        return deleteAcls(filters, new DeleteAclsOptions());
    }

    /**
     * Deletes access control lists (ACLs) according to the supplied filters.
     *
     * This operation is not transactional so it may succeed for some ACLs while fail for others.
     *
     * This operation is supported by brokers with version 0.11.0.0 or higher.
     *
     * @param filters           The filters to use.
     * @param options           The options to use when deleting the ACLs.
     * @return                  The DeleteAclsResult.
     */
    DeleteAclsResult deleteAcls(Collection<AclBindingFilter> filters, DeleteAclsOptions options);


    /**
     * Get the configuration for the specified resources with the default options.
     *
     * This is a convenience method for #{@link Admin#describeConfigs(Collection, DescribeConfigsOptions)} with
default options.
     * See the overload for more details.
     *
     * This operation is supported by brokers with version 0.11.0.0 or higher.
     *
     * @param resources         The resources (topic and broker resource types are currently supported)
     * @return                  The DescribeConfigsResult
     */
    default DescribeConfigsResult describeConfigs(Collection<ConfigResource> resources) {
        return describeConfigs(resources, new DescribeConfigsOptions());
    }

    /**
     * Get the configuration for the specified resources.
     *
     * The returned configuration includes default values and the isDefault() method can be used to distinguish
them
     * from user supplied values.
     *
     * The value of config entries where isSensitive() is true is always {@code null} so that sensitive
information
     * is not disclosed.
     *
     * Config entries where isReadOnly() is true cannot be updated.
     *
     * This operation is supported by brokers with version 0.11.0.0 or higher.
     *
     * @param resources         The resources (topic and broker resource types are currently supported)
     * @param options           The options to use when describing configs
     * @return                  The DescribeConfigsResult
     */
    DescribeConfigsResult describeConfigs(Collection<ConfigResource> resources, DescribeConfigsOptions options);

    /**
     * Update the configuration for the specified resources with the default options.
     *
     * This is a convenience method for #{@link Admin#alterConfigs(Map, AlterConfigsOptions)} with default
```

```
options.
     * See the overload for more details.
     *
     * This operation is supported by brokers with version 0.11.0.0 or higher.
     *
     * @param configs         The resources with their configs (topic is the only resource type with configs
that can
     *                        be updated currently)
     * @return                The AlterConfigsResult
     * @deprecated Since 2.3. Use {@link #incrementalAlterConfigs(Map)}.
     */
    @Deprecated
    default AlterConfigsResult alterConfigs(Map<ConfigResource, Config> configs) {
        return alterConfigs(configs, new AlterConfigsOptions());
    }

    /**
     * Update the configuration for the specified resources with the default options.
     *
     * Updates are not transactional so they may succeed for some resources while fail for others. The configs
for
     * a particular resource are updated atomically.
     *
     * This operation is supported by brokers with version 0.11.0.0 or higher.
     *
     * @param configs         The resources with their configs (topic is the only resource type with configs
that can
     *                        be updated currently)
     * @param options         The options to use when describing configs
     * @return                The AlterConfigsResult
     * @deprecated Since 2.3. Use {@link #incrementalAlterConfigs(Map, AlterConfigsOptions)}.
     */
    @Deprecated
    AlterConfigsResult alterConfigs(Map<ConfigResource, Config> configs, AlterConfigsOptions options);

    /**
     * Incrementally updates the configuration for the specified resources with default options.
     *
     * This is a convenience method for #{@link Admin#incrementalAlterConfigs(Map, AlterConfigsOptions)} with
default options.
     * See the overload for more details.*
     *
     * This operation is supported by brokers with version 2.3.0 or higher.
     *
     * @param configs         The resources with their configs
     * @return                The IncrementalAlterConfigsResult
     */
    default AlterConfigsResult incrementalAlterConfigs(Map<ConfigResource, Collection<AlterConfigOp>> configs) {
        return incrementalAlterConfigs(configs, new AlterConfigsOptions());
    }


    /**
     * Incrementally update the configuration for the specified resources.
     *
     * Updates are not transactional so they may succeed for some resources while fail for others. The configs
for
     * a particular resource are updated atomically.
     *
     * <p>The following exceptions can be anticipated when calling {@code get()} on the futures obtained from
     * the returned {@code IncrementalAlterConfigsResult}:</p>
     * <ul>
     *   <li>{@link org.apache.kafka.common.errors.ClusterAuthorizationException}
     *   if the authenticated user didn't have alter access to the cluster.</li>
     *   <li>{@link org.apache.kafka.common.errors.TopicAuthorizationException}
     *   if the authenticated user didn't have alter access to the Topic.</li>
     *   <li>{@link org.apache.kafka.common.errors.InvalidRequestException}
     *   if the request details are invalid. e.g., a configuration key was specified more than once for a
resource</li>
     * </ul>*
     *
```

```
     * This operation is supported by brokers with version 2.3.0 or higher.
     *
     * @param configs         The resources with their configs
     * @param options         The options to use when altering configs
     * @return                The IncrementalAlterConfigsResult
     */
    AlterConfigsResult incrementalAlterConfigs(Map<ConfigResource,
        Collection<AlterConfigOp>> configs, AlterConfigsOptions options);

    /**
     * Change the log directory for the specified replicas. If the replica does not exist on the broker, the
result
     * shows REPLICA_NOT_AVAILABLE for the given replica and the replica will be created in the given log
directory on the
     * broker when it is created later. If the replica already exists on the broker, the replica will be moved
to the given
     * log directory if it is not already there.
     *
     * This operation is not transactional so it may succeed for some replicas while fail for others.
     *
     * This is a convenience method for #{@link Admin#alterReplicaLogDirs(Map, AlterReplicaLogDirsOptions)}
with default options.
     * See the overload for more details.
     *
     * This operation is supported by brokers with version 1.1.0 or higher.
     *
     * @param replicaAssignment  The replicas with their log directory absolute path
     * @return                   The AlterReplicaLogDirsResult
     */
    default AlterReplicaLogDirsResult alterReplicaLogDirs(Map<TopicPartitionReplica, String> replicaAssignment)
{
        return alterReplicaLogDirs(replicaAssignment, new AlterReplicaLogDirsOptions());
    }

    /**
     * Change the log directory for the specified replicas. If the replica does not exist on the broker, the
result
     * shows REPLICA_NOT_AVAILABLE for the given replica and the replica will be created in the given log
directory on the
     * broker when it is created later. If the replica already exists on the broker, the replica will be moved
to the given
     * log directory if it is not already there.
     *
     * This operation is not transactional so it may succeed for some replicas while fail for others.
     *
     * This operation is supported by brokers with version 1.1.0 or higher.
     *
     * @param replicaAssignment  The replicas with their log directory absolute path
     * @param options            The options to use when changing replica dir
     * @return                   The AlterReplicaLogDirsResult
     */
    AlterReplicaLogDirsResult alterReplicaLogDirs(Map<TopicPartitionReplica, String> replicaAssignment,
                                                  AlterReplicaLogDirsOptions options);

    /**
     * Query the information of all log directories on the given set of brokers
     *
     * This is a convenience method for #{@link Admin#describeLogDirs(Collection, DescribeLogDirsOptions)} with
default options.
     * See the overload for more details.
     *
     * This operation is supported by brokers with version 1.0.0 or higher.
     *
     * @param brokers     A list of brokers
     * @return            The DescribeLogDirsResult
     */
    default DescribeLogDirsResult describeLogDirs(Collection<Integer> brokers) {
        return describeLogDirs(brokers, new DescribeLogDirsOptions());
    }

    /**
```

```
     * Query the information of all log directories on the given set of brokers
     *
     * This operation is supported by brokers with version 1.0.0 or higher.
     *
     * @param brokers     A list of brokers
     * @param options     The options to use when querying log dir info
     * @return            The DescribeLogDirsResult
     */
    DescribeLogDirsResult describeLogDirs(Collection<Integer> brokers, DescribeLogDirsOptions options);

    /**
     * Query the replica log directory information for the specified replicas.
     *
     * This is a convenience method for #{@link Admin#describeReplicaLogDirs(Collection,
DescribeReplicaLogDirsOptions)}
     * with default options. See the overload for more details.
     *
     * This operation is supported by brokers with version 1.0.0 or higher.
     *
     * @param replicas      The replicas to query
     * @return              The DescribeReplicaLogDirsResult
     */
    default DescribeReplicaLogDirsResult describeReplicaLogDirs(Collection<TopicPartitionReplica> replicas) {
        return describeReplicaLogDirs(replicas, new DescribeReplicaLogDirsOptions());
    }

    /**
     * Query the replica log directory information for the specified replicas.
     *
     * This operation is supported by brokers with version 1.0.0 or higher.
     *
     * @param replicas      The replicas to query
     * @param options       The options to use when querying replica log dir info
     * @return              The DescribeReplicaLogDirsResult
     */
    DescribeReplicaLogDirsResult describeReplicaLogDirs(Collection<TopicPartitionReplica> replicas,
DescribeReplicaLogDirsOptions options);

    /**
     * <p>Increase the number of partitions of the topics given as the keys of {@code newPartitions}
     * according to the corresponding values. <strong>If partitions are increased for a topic that has a key,
     * the partition logic or ordering of the messages will be affected.</strong></p>
     *
     * <p>This is a convenience method for {@link #createPartitions(Map, CreatePartitionsOptions)} with default
options.
     * See the overload for more details.</p>
     *
     * @param newPartitions The topics which should have new partitions created, and corresponding parameters
     *                      for the created partitions.
     * @return              The CreatePartitionsResult.
     */
    default CreatePartitionsResult createPartitions(Map<String, NewPartitions> newPartitions) {
        return createPartitions(newPartitions, new CreatePartitionsOptions());
    }

    /**
     * <p>Increase the number of partitions of the topics given as the keys of {@code newPartitions}
     * according to the corresponding values. <strong>If partitions are increased for a topic that has a key,
     * the partition logic or ordering of the messages will be affected.</strong></p>
     *
     * <p>This operation is not transactional so it may succeed for some topics while fail for others.</p>
     *
     * <p>It may take several seconds after this method returns
     * success for all the brokers to become aware that the partitions have been created.
     * During this time, {@link Admin#describeTopics(Collection)}
     * may not return information about the new partitions.</p>
     *
     * <p>This operation is supported by brokers with version 1.0.0 or higher.</p>
     *
     * <p>The following exceptions can be anticipated when calling {@code get()} on the futures obtained from
the
```

```
     * {@link CreatePartitionsResult#values() values()} method of the returned {@code CreatePartitionsResult}<
/p>
     * <ul>
     *     <li>{@link org.apache.kafka.common.errors.AuthorizationException}
     *     if the authenticated user is not authorized to alter the topic</li>
     *     <li>{@link org.apache.kafka.common.errors.TimeoutException}
     *     if the request was not completed in within the given {@link CreatePartitionsOptions#timeoutMs()}.<
/li>
     *     <li>{@link org.apache.kafka.common.errors.ReassignmentInProgressException}
     *     if a partition reassignment is currently in progress</li>
     *     <li>{@link org.apache.kafka.common.errors.BrokerNotAvailableException}
     *     if the requested {@link NewPartitions#assignments()} contain a broker that is currently unavailable.<
/li>
     *     <li>{@link org.apache.kafka.common.errors.InvalidReplicationFactorException}
     *     if no {@link NewPartitions#assignments()} are given and it is impossible for the broker to assign
     *     replicas with the topics replication factor.</li>
     *     <li>Subclasses of {@link org.apache.kafka.common.KafkaException}
     *     if the request is invalid in some way.</li>
     * </ul>
     *
     * @param newPartitions The topics which should have new partitions created, and corresponding parameters
     *                      for the created partitions.
     * @param options       The options to use when creating the new paritions.
     * @return              The CreatePartitionsResult.
     */
    CreatePartitionsResult createPartitions(Map<String, NewPartitions> newPartitions,
                                            CreatePartitionsOptions options);

    /**
     * Delete records whose offset is smaller than the given offset of the corresponding partition.
     *
     * This is a convenience method for {@link #deleteRecords(Map, DeleteRecordsOptions)} with default options.
     * See the overload for more details.
     *
     * This operation is supported by brokers with version 0.11.0.0 or higher.
     *
     * @param recordsToDelete       The topic partitions and related offsets from which records deletion starts.
     * @return                      The DeleteRecordsResult.
     */
    default DeleteRecordsResult deleteRecords(Map<TopicPartition, RecordsToDelete> recordsToDelete) {
        return deleteRecords(recordsToDelete, new DeleteRecordsOptions());
    }

    /**
     * Delete records whose offset is smaller than the given offset of the corresponding partition.
     *
     * This operation is supported by brokers with version 0.11.0.0 or higher.
     *
     * @param recordsToDelete       The topic partitions and related offsets from which records deletion starts.
     * @param options               The options to use when deleting records.
     * @return                      The DeleteRecordsResult.
     */
    DeleteRecordsResult deleteRecords(Map<TopicPartition, RecordsToDelete> recordsToDelete,
                                      DeleteRecordsOptions options);

    /**
     * <p>Create a Delegation Token.</p>
     *
     * <p>This is a convenience method for {@link #createDelegationToken(CreateDelegationTokenOptions)} with
default options.
     * See the overload for more details.</p>
     *
     * @return                      The CreateDelegationTokenResult.
     */
    default CreateDelegationTokenResult createDelegationToken() {
        return createDelegationToken(new CreateDelegationTokenOptions());
    }


    /**
     * <p>Create a Delegation Token.</p>
```

```
     *
     * <p>This operation is supported by brokers with version 1.1.0 or higher.</p>
     *
     * <p>The following exceptions can be anticipated when calling {@code get()} on the futures obtained from
the
     * {@link CreateDelegationTokenResult#delegationToken() delegationToken()} method of the returned {@code
CreateDelegationTokenResult}</p>
     * <ul>
     *     <li>{@link org.apache.kafka.common.errors.UnsupportedByAuthenticationException}
     *     If the request sent on PLAINTEXT/1-way SSL channels or delegation token authenticated channels.</li>
     *     <li>{@link org.apache.kafka.common.errors.InvalidPrincipalTypeException}
     *     if the renewers principal type is not supported.</li>
     *     <li>{@link org.apache.kafka.common.errors.DelegationTokenDisabledException}
     *     if the delegation token feature is disabled.</li>
     *     <li>{@link org.apache.kafka.common.errors.TimeoutException}
     *     if the request was not completed in within the given {@link
CreateDelegationTokenOptions#timeoutMs()}.</li>
     * </ul>
     *
     * @param options               The options to use when creating delegation token.
     * @return                      The DeleteRecordsResult.
     */
    CreateDelegationTokenResult createDelegationToken(CreateDelegationTokenOptions options);


    /**
     * <p>Renew a Delegation Token.</p>
     *
     * <p>This is a convenience method for {@link #renewDelegationToken(byte[], RenewDelegationTokenOptions)}
with default options.
     * See the overload for more details.</p>
     *
     *
     * @param hmac                  HMAC of the Delegation token
     * @return                      The RenewDelegationTokenResult.
     */
    default RenewDelegationTokenResult renewDelegationToken(byte[] hmac) {
        return renewDelegationToken(hmac, new RenewDelegationTokenOptions());
    }

    /**
     * <p> Renew a Delegation Token.</p>
     *
     * <p>This operation is supported by brokers with version 1.1.0 or higher.</p>
     *
     * <p>The following exceptions can be anticipated when calling {@code get()} on the futures obtained from
the
     * {@link RenewDelegationTokenResult#expiryTimestamp() expiryTimestamp()} method of the returned {@code
RenewDelegationTokenResult}</p>
     * <ul>
     *     <li>{@link org.apache.kafka.common.errors.UnsupportedByAuthenticationException}
     *     If the request sent on PLAINTEXT/1-way SSL channels or delegation token authenticated channels.</li>
     *     <li>{@link org.apache.kafka.common.errors.DelegationTokenDisabledException}
     *     if the delegation token feature is disabled.</li>
     *     <li>{@link org.apache.kafka.common.errors.DelegationTokenNotFoundException}
     *     if the delegation token is not found on server.</li>
     *     <li>{@link org.apache.kafka.common.errors.DelegationTokenOwnerMismatchException}
     *     if the authenticated user is not owner/renewer of the token.</li>
     *     <li>{@link org.apache.kafka.common.errors.DelegationTokenExpiredException}
     *     if the delegation token is expired.</li>
     *     <li>{@link org.apache.kafka.common.errors.TimeoutException}
     *     if the request was not completed in within the given {@link RenewDelegationTokenOptions#timeoutMs()}.
</li>
     * </ul>
     *
     * @param hmac                  HMAC of the Delegation token
     * @param options               The options to use when renewing delegation token.
     * @return                      The RenewDelegationTokenResult.
     */
    RenewDelegationTokenResult renewDelegationToken(byte[] hmac, RenewDelegationTokenOptions options);
```

```
    /**
     * <p>Expire a Delegation Token.</p>
     *
     * <p>This is a convenience method for {@link #expireDelegationToken(byte[], ExpireDelegationTokenOptions)}
with default options.
     * This will expire the token immediately. See the overload for more details.</p>
     *
     * @param hmac                  HMAC of the Delegation token
     * @return                      The ExpireDelegationTokenResult.
     */
    default ExpireDelegationTokenResult expireDelegationToken(byte[] hmac) {
        return expireDelegationToken(hmac, new ExpireDelegationTokenOptions());
    }

    /**
     * <p>Expire a Delegation Token.</p>
     *
     * <p>This operation is supported by brokers with version 1.1.0 or higher.</p>
     *
     * <p>The following exceptions can be anticipated when calling {@code get()} on the futures obtained from
the
     * {@link ExpireDelegationTokenResult#expiryTimestamp() expiryTimestamp()} method of the returned {@code
ExpireDelegationTokenResult}</p>
     * <ul>
     *     <li>{@link org.apache.kafka.common.errors.UnsupportedByAuthenticationException}
     *     If the request sent on PLAINTEXT/1-way SSL channels or delegation token authenticated channels.</li>
     *     <li>{@link org.apache.kafka.common.errors.DelegationTokenDisabledException}
     *     if the delegation token feature is disabled.</li>
     *     <li>{@link org.apache.kafka.common.errors.DelegationTokenNotFoundException}
     *     if the delegation token is not found on server.</li>
     *     <li>{@link org.apache.kafka.common.errors.DelegationTokenOwnerMismatchException}
     *     if the authenticated user is not owner/renewer of the requested token.</li>
     *     <li>{@link org.apache.kafka.common.errors.DelegationTokenExpiredException}
     *     if the delegation token is expired.</li>
     *     <li>{@link org.apache.kafka.common.errors.TimeoutException}
     *     if the request was not completed in within the given {@link
ExpireDelegationTokenOptions#timeoutMs()}.</li>
     * </ul>
     *
     * @param hmac                  HMAC of the Delegation token
     * @param options               The options to use when expiring delegation token.
     * @return                      The ExpireDelegationTokenResult.
     */
    ExpireDelegationTokenResult expireDelegationToken(byte[] hmac, ExpireDelegationTokenOptions options);

    /**
     *<p>Describe the Delegation Tokens.</p>
     *
     * <p>This is a convenience method for {@link #describeDelegationToken(DescribeDelegationTokenOptions)}
with default options.
     * This will return all the user owned tokens and tokens where user have Describe permission. See the
overload for more details.</p>
     *
     * @return                      The DescribeDelegationTokenResult.
     */
    default DescribeDelegationTokenResult describeDelegationToken() {
        return describeDelegationToken(new DescribeDelegationTokenOptions());
    }

    /**
     * <p>Describe the Delegation Tokens.</p>
     *
     * <p>This operation is supported by brokers with version 1.1.0 or higher.</p>
     *
     * <p>The following exceptions can be anticipated when calling {@code get()} on the futures obtained from
the
     * {@link DescribeDelegationTokenResult#delegationTokens() delegationTokens()} method of the returned
{@code DescribeDelegationTokenResult}</p>
     * <ul>
     *     <li>{@link org.apache.kafka.common.errors.UnsupportedByAuthenticationException}
     *     If the request sent on PLAINTEXT/1-way SSL channels or delegation token authenticated channels.</li>
```

```
     *     <li>{@link org.apache.kafka.common.errors.DelegationTokenDisabledException}
     *     if the delegation token feature is disabled.</li>
     *     <li>{@link org.apache.kafka.common.errors.TimeoutException}
     *     if the request was not completed in within the given {@link
DescribeDelegationTokenOptions#timeoutMs()}.</li>
     * </ul>
     *
     * @param options               The options to use when describing delegation tokens.
     * @return                      The DescribeDelegationTokenResult.
     */
    DescribeDelegationTokenResult describeDelegationToken(DescribeDelegationTokenOptions options);

    /**
     * Describe some group IDs in the cluster.
     *
     * @param groupIds The IDs of the groups to describe.
     * @param options  The options to use when describing the groups.
     * @return The DescribeConsumerGroupResult.
     */
    DescribeConsumerGroupsResult describeConsumerGroups(Collection<String> groupIds,
                                                        DescribeConsumerGroupsOptions options);

    /**
     * Describe some group IDs in the cluster, with the default options.
     * <p>
     * This is a convenience method for
     * #{@link Admin#describeConsumerGroups(Collection, DescribeConsumerGroupsOptions)} with
     * default options. See the overload for more details.
     *
     * @param groupIds The IDs of the groups to describe.
     * @return The DescribeConsumerGroupResult.
     */
    default DescribeConsumerGroupsResult describeConsumerGroups(Collection<String> groupIds) {
        return describeConsumerGroups(groupIds, new DescribeConsumerGroupsOptions());
    }

    /**
     * List the consumer groups available in the cluster.
     *
     * @param options           The options to use when listing the consumer groups.
     * @return The ListGroupsResult.
     */
    ListConsumerGroupsResult listConsumerGroups(ListConsumerGroupsOptions options);

    /**
     * List the consumer groups available in the cluster with the default options.
     *
     * This is a convenience method for #{@link Admin#listConsumerGroups(ListConsumerGroupsOptions)} with
default options.
     * See the overload for more details.
     *
     * @return The ListGroupsResult.
     */
    default ListConsumerGroupsResult listConsumerGroups() {
        return listConsumerGroups(new ListConsumerGroupsOptions());
    }

    /**
     * List the consumer group offsets available in the cluster.
     *
     * @param options           The options to use when listing the consumer group offsets.
     * @return The ListGroupOffsetsResult
     */
    ListConsumerGroupOffsetsResult listConsumerGroupOffsets(String groupId, ListConsumerGroupOffsetsOptions
options);

    /**
     * List the consumer group offsets available in the cluster with the default options.
     *
     * This is a convenience method for #{@link Admin#listConsumerGroupOffsets(String,
ListConsumerGroupOffsetsOptions)} with default options.
```

```
     *
     * @return The ListGroupOffsetsResult.
     */
    default ListConsumerGroupOffsetsResult listConsumerGroupOffsets(String groupId) {
        return listConsumerGroupOffsets(groupId, new ListConsumerGroupOffsetsOptions());
    }

    /**
     * Delete consumer groups from the cluster.
     *
     * @param options           The options to use when deleting a consumer group.
     * @return The DeletConsumerGroupResult.
     */
    DeleteConsumerGroupsResult deleteConsumerGroups(Collection<String> groupIds, DeleteConsumerGroupsOptions
options);

    /**
     * Delete consumer groups from the cluster with the default options.
     *
     * @return The DeleteConsumerGroupResult.
     */
    default DeleteConsumerGroupsResult deleteConsumerGroups(Collection<String> groupIds) {
        return deleteConsumerGroups(groupIds, new DeleteConsumerGroupsOptions());
    }

    /**
     * Elect the preferred replica as leader for topic partitions.
     *
     * This is a convenience method for {@link #electLeaders(ElectionType, Set, ElectLeadersOptions)}
     * with preferred election type and default options.
     *
     * This operation is supported by brokers with version 2.2.0 or higher.
     *
     * @param partitions      The partitions for which the preferred leader should be elected.
     * @return                The ElectPreferredLeadersResult.
     * @deprecated            Since 2.4.0. Use {@link #electLeaders(ElectionType, Set)}.
     */
    @Deprecated
    default ElectPreferredLeadersResult electPreferredLeaders(Collection<TopicPartition> partitions) {
        return electPreferredLeaders(partitions, new ElectPreferredLeadersOptions());
    }

    /**
     * Elect the preferred replica as leader for topic partitions.
     *
     * This is a convenience method for {@link #electLeaders(ElectionType, Set, ElectLeadersOptions)}
     * with preferred election type.
     *
     * This operation is supported by brokers with version 2.2.0 or higher.
     *
     * @param partitions      The partitions for which the preferred leader should be elected.
     * @param options         The options to use when electing the preferred leaders.
     * @return                The ElectPreferredLeadersResult.
     * @deprecated            Since 2.4.0. Use {@link #electLeaders(ElectionType, Set, ElectLeadersOptions)}.
     */
    @Deprecated
    default ElectPreferredLeadersResult electPreferredLeaders(Collection<TopicPartition> partitions,
                                                              ElectPreferredLeadersOptions options) {
        final ElectLeadersOptions newOptions = new ElectLeadersOptions();
        newOptions.timeoutMs(options.timeoutMs());
        final Set<TopicPartition> topicPartitions = partitions == null ? null : new HashSet<>(partitions);

        return new ElectPreferredLeadersResult(electLeaders(ElectionType.PREFERRED, topicPartitions,
newOptions));
    }

    /**
     * Elect a replica as leader for topic partitions.
     *
     * This is a convenience method for {@link #electLeaders(ElectionType, Set, ElectLeadersOptions)}
     * with default options.
```

```
     *
     * @param electionType            The type of election to conduct.
     * @param partitions              The topics and partitions for which to conduct elections.
     * @return                        The ElectLeadersResult.
     */
    default ElectLeadersResult electLeaders(ElectionType electionType, Set<TopicPartition> partitions) {
        return electLeaders(electionType, partitions, new ElectLeadersOptions());
    }

    /**
     * Elect a replica as leader for the given {@code partitions}, or for all partitions if the argumentl
     * to {@code partitions} is null.
     *
     * This operation is not transactional so it may succeed for some partitions while fail for others.
     *
     * It may take several seconds after this method returns success for all the brokers in the cluster
     * to become aware that the partitions have new leaders. During this time,
     * {@link Admin#describeTopics(Collection)} may not return information about the partitions'
     * new leaders.
     *
     * This operation is supported by brokers with version 2.2.0 or later if preferred eleciton is use;
     * otherwise the brokers most be 2.4.0 or higher.
     *
     * <p>The following exceptions can be anticipated when calling {@code get()} on the future obtained
     * from the returned {@code ElectLeadersResult}:</p>
     * <ul>
     *   <li>{@link org.apache.kafka.common.errors.ClusterAuthorizationException}
     *   if the authenticated user didn't have alter access to the cluster.</li>
     *   <li>{@link org.apache.kafka.common.errors.UnknownTopicOrPartitionException}
     *   if the topic or partition did not exist within the cluster.</li>
     *   <li>{@link org.apache.kafka.common.errors.InvalidTopicException}
     *   if the topic was already queued for deletion.</li>
     *   <li>{@link org.apache.kafka.common.errors.NotControllerException}
     *   if the request was sent to a broker that was not the controller for the cluster.</li>
     *   <li>{@link org.apache.kafka.common.errors.TimeoutException}
     *   if the request timed out before the election was complete.</li>
     *   <li>{@link org.apache.kafka.common.errors.LeaderNotAvailableException}
     *   if the preferred leader was not alive or not in the ISR.</li>
     * </ul>
     *
     * @param electionType            The type of election to conduct.
     * @param partitions              The topics and partitions for which to conduct elections.
     * @param options                 The options to use when electing the leaders.
     * @return                        The ElectLeadersResult.
     */
    ElectLeadersResult electLeaders(
            ElectionType electionType,
            Set<TopicPartition> partitions,
            ElectLeadersOptions options);

    /**
     * Get the metrics kept by the adminClient
     */
    Map<MetricName, ? extends Metric> metrics();
}
```

Kafka Stream's KafkaClientSupplier

```
public interface KafkaClientSupplier {
    /**
     * Create an {@link AdminClient} which is used for internal topic management.
     *
     * @param config Supplied by the {@link java.util.Properties} given to the {@link KafkaStreams}
     * @return an instance of {@link AdminClient}
     * @deprecated Not called by Kafka Streams, which now uses {@link #getAdmin} instead.
     */
    @Deprecated
    default AdminClient getAdminClient(final Map<String, Object> config) {
        throw new UnsupportedOperationException("Direct use of this method is deprecated. " +
            "Implementations of KafkaClientSupplier should implement the getAdmin method instead. " +
            "The method will be removed in a future release.");
    }

    /**
     * Create an {@link Admin} which is used for internal topic management.
     *
     * @param config Supplied by the {@link java.util.Properties} given to the {@link KafkaStreams}
     * @return an instance of {@link Admin}
     */
    @SuppressWarnings("deprecation")
    default Admin getAdmin(final Map<String, Object> config) {
        return getAdminClient(config);
    }

    ... rest of methods remain the same.
}
```

Proposed Changes

As per section above.

Compatibility, Deprecation, and Migration Plan

Adding the interface and moving the default implementations of methods to the interface is a binary compatible change, so will not affect existing clients.

Rejected Alternatives

Deprecation

It was originally suggested to also deprecate the now empty `AdminClient` abstract class. However, it was felt that this would cause too much unnecessary noise for downstream users of the library, especially for uses where multiple versions of Kafka are supported, e.g. Flink, Spark, where the warnings would be unavoidable and the user can do nothing but disable them.