# KIP-422: Add support for client quota configuration in the Kafka Admin Client



## Status

**Current state**: *Rejected in favor of KIP-546*

**Discussion thread**: here

**JIRA**: *KAFKA-7740*

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

## Motivation

Currently, the AdminClient supports a long list of APIs that can be leveraged by users in many different ways. For instance, the AdminClient allows users to manage entities such as topics, partitions, consumer offset, ACLs, replication log directories and so on. In these scenarios, users only need talk to brokers through AdminClient and there is no direct dependency between client applications and Zookeeper any more.

One missing piece in the AdminClient is the configuration of client quotas. Without this feature, users have to directly talk to Zookeeper in order to manage the quota configurations. This KIP is designed to bring this feature into the AdminClient library.

# Public Interfaces

**1.The following quota related APIs will be added to the AdminClient from package org.apache.kafka.clients.admin.**

```
/**
* This is a convenient method for #{@link AdminClient#setQuotas(Map, SetQuotasOptions)} with default options.
*
* This operation is not transactional so it may succeed for some quotas while fail for others.
*
* This operation is supported by brokers with version 0.11.0.0 or higher.
*
* @param quotas The quotas to use.
* @return The SetQuotaResults.
*/
public SetQuotasResults setQuotas(Map<QuotaTarget, QuotaLimit> quotas) {
return setQuotas(quotas, new SetQuotasOptions());
}

/**
* Set quotas for the specified Quota Targets.
*
* This operation is not transactional so it may succeed for some quotas while fail for others.
*
* This operation is supported by brokers with version 0.11.0.0 or higher.
*
* @param quotas The quotas to use.
* @param options The options to use when set the quotas.
* @return The SetQuotaResults.
*/
public abstract SetQuotasResults setQuotas(Map<QuotaTarget, QuotaLimit> quotas, SetQuotasOptions options);

/**
* This is a convenient method for #{@link AdminClient#describeQuota(Map, DescribeQuotasOptions)} with default options.
* Get the quota settings for the specific Quota Target.
*
* @param target The quota target to use.
* @return The DescribeQuotasResults.
*/
public DescribeQuotasResults describeQuota(QuotaTarget target) {
return describeQuota(target, new DescribeQuotasOptions());
}

/**
* Get the quota settings for the specific Quota Target.
*
* @param target The quota target to use.
* @param options The options to use when describe the quotas.
* @return The DescribeQuotasResults.
*/
public abstract DescribeQuotasResults describeQuota(QuotaTarget target, DescribeQuotasOptions options);

/**
* Delete the quota setting for the specified quota target.
* @param quotaTargets The quota target to use.
* @param options The options to use when delete the quota.
* @return The DeleteQuotasResults
*/
public abstract DeleteQuotasResults deleteQuotas(Collection<QuotaTarget> quotaTargets, DeleteQuotasOptions options);

/**
* Delete the quota setting for the specified quota target.
* @param quotaTargets The quota target to use.
* @return The DeleteQuotasResults
*/
public DeleteQuotasResults deleteQuotas(Collection<QuotaTarget> quotaTargets) {
return deleteQuotas(quotaTargets, new DeleteQuotasOptions());
}
```

**2. Add the QuotaTarget interface to represent the identities of quota targets, it is implemented by ClientQuotaTarget, UserQuotaTarget and UerAndClientIdQuotaTarget types.**

```
package org.apache.kafka.server.quota;

public interface QuotaTarget {
/**
* Return the quota target type.
*/
QuotaTargetType getQuotaTargetType();
}
```

**3. Add the QuotaLimit objects to represent different quota configurations, e.g. produce_rate = 10000 bytes/s**

```
package org.apache.kafka.server.quota;

public class QuotaLimit {
    private long quotaValue;
    private ClientQuotaType quotaType;

    public QuotaLimit(ClientQuotaType quotaType, long quotaValue) {
        this.quotaType = quotaType;
        this.quotaValue = quotaValue;
    }

    /**
     * Get the quota value for this Quota limit.
     *
     * @return      For ClientQuotaType.PRODUCE and ClientQuotaType.FETCH, return the quota value in unit of
bytesPerSec.
     *              For ClientQuotaType.REQUEST, return the percentage value.
     */
    public long getQuotaValue() {
        return quotaValue;
    }

    /**
     * Get the quota type.
     * @return the ClientQuotaType.
     */
    public ClientQuotaType getQuotaType() {
        return quotaType;
    }
}
```

# Proposed Changes

The existing protocols will be used to send requests from AdminClient to any broker for the quota management, because fundamentally the quota management is one category of the general configuration changes

To be specific, the following request/response pairs will be leveraged:

1. IncrementalAlterConfigsRequest and IncrementalAlterConfigsResponse
2. DescribeConfigsRequest and DescribeConfigsResponse

The AdminClient will convert the quota management requests from the client side to the Configuration alter/describe requests, before sending them to Kafka brokers.

USER and CLIENT resources need to be added in the ConfigResource.java interface**.** They represent the targeting entities for the quota management.

```
public final class ConfigResource {

 public enum Type {
        BROKER((byte) 4), USER((byte) 3), TOPIC((byte) 2), CLIENT((byte) 1), UNKNOWN((byte) 0); // we add USER
and CLIENT type.
    ...
 }

}
```

Also we would like to dynamically config user and client entities.

```
/**
 * Source of configuration entries.
 */
public enum ConfigSource {

  ...

 DYNAMIC_USER_CONFIG, // dynamic user config that is configured for a specific user.
 DYNAMIC_CLIENT_CONFIG, // dynamic client config that is configured for a specific client.
  ...
}
```

In the KafkaApi class, add the user and client configuration processing logics in the handleIncrementalAlterConfigsRequest and handleDescribeConfigsRequest.

Internally, they will call the adminZkClient, in order to read/write configurations from/to Zookeeper, and then return the results to users.

The ACL for the quota management is in the Cluster level.

# Compatibility, Deprecation, and Migration Plan

There is no known compatibility issue introduced in this KIP.

There is also no feature to deprecate from this KIP.

No need to migration, as data are always in the Zookeeper.

# Rejected Alternatives

Manage the quotas through existing configuration management APIs.

1. Expose the lower level quota implementation details to users.
2. Inconvenient for users to manage the quota. Users have to construct the right property name for each quota types, and it's error prone.
3. Hard to extend in the future.