KIP-140: Add administrative RPCs for adding, deleting, and listing ACLs

Contents

Status

Current state:accepted

Discussion thread: https://www.mail-archive.com/dev@kafka.apache.org/msg70858.html

JIRA: ⚠ Unable to render Jira issues macro, execution error.

Released: 0.11.0

Motivation

As part of the KIP-117 work to create an **AdminClient** for Kafka, we would like to have a way of adding, deleting, and listing the access control lists (ACLs) which are used to control access on Kafka topics and brokers.

New Network Requests and Responses

Representing ACL Components on the Wire

Each ACL consists of a 4-tuple of (principal, host, operation, permission_type).

The "principal" may be either a specific string that matches a single principal, or a "wildcard" represented by the string "*". In the wire protocol, we represent principal as a NULLABLE_STRING.

The "host" is the IP address which the ACL applies to, or "*" if the ACL applies to all IP addresses. In the wire protocol, we represent host as a NULLABLE_STRING.

The "operation" is the particular operation which the ACL controls. In the wire protocol, we represented resource_type as an INT8. The mappings are:

- `0: unknown`

- 1: any
- 2: all
- 3: read
- 4: write
- 5: create
- 6: delete
- 7: alter
- 8: describe
- 9: clusterAction

"Unknown" represents an operation type that we don't know how to decode. "Any" can only be used in filters, and matches any operation type.

The "permission_type" is whether the ACL allows access or forbids it. In the wire protocol, we represented resource_type as an INT8. The mappings are:

- 0: unknown
- 1: any
- 2: deny
- 3: allow

"Unknown" represents a permission type that we don't know how to decode. "Any" can only be used in filters, and matches any permission type.

Representing Resource Components on the Wire

Every ACLs is bound to a specific resource.

The "resource_type" is the type of resource the ACL applies to. In the wire protocol, we represented resource_type as an INT8. The mappings are:

- 0: unknown
- 1: any
- 2: topic
- 3: group
- 4: cluster
- 5: transactional_id

"Unknown" represents a resource type that we don't know how to decode. "Any" can only be used in filters, and matches any resource type.

The "resource_name" is the name of the particular resource. For example, when "resource_type" == "topic", "resource_name" will be the topic name. In the wire protocol, we represent principal as a NULLABLE_STRING.

DescribeAclsRequest and DescribeAclsResponse

DescribeAclsRequest handles describing the ACLs in the cluster. Principals must possess Cluster:Describe permissions to call DescribeAclsRequest, or be superuser. Unauthorized requests will receive a ClusterAuthorizationException.

```
DescribeAclsRequest (Version: 0) => principal host operation permission_type resource_type resource_name
  principal => NULLABLE_STRING
  host => NULLABLE_STRING
  operation => INT8
  permission_type => INT8
  resource_type => INT8
  resource_name => NULLABLE_STRING
```

The arguments to DescribeAclsRequest are ANDed together to act as a filter. For example, if a principal is supplied, we will return only ACLs that match that principal. If an operation is supplied, we will return only ACLs that include that operation. And so forth. This capability can be used to easily list all the ACLs that apply to a particular topic, or a particular principal.

Note that an argument of "any" or null is different than a wildcard argument. That is, DescribeAclsRequest(principal=null) will return all ACLs, but DescribeAclsRequest(principal=*) will return only ACLs that have their principal set to wildcard.

```
DescribeAclsResponse (Version: 0) => error_code error_message [resource]
  error_code => INT16
  error_message => NULLABLE_STRING
  resource => resource_type resource_name [acl]
    resource_type => INT8
    resource_name => STRING
    acl => principal host operation permission_type
      principal => STRING
      host => STRING
      operation => INT8
      permission_type => INT8
```

The error_code field will be non-zero if there was an error processing the request. If the error_code is non-zero, the results list will be empty. Otherwise, each listed resource object describes a specific resource, and the ACLs bound to that resource. Note that if filters were specified in the DescribeAclsRequest, this may not be a complete list of all the ACLs bound to the resource, but only the ones which matched the supplied filters. None of the fields in the ACL 4-tuple or the resource 2-tuple are ever set to null or none in the response.

CreateAclsRequest and CreateAclsResponse

CreateAclsRequest handles adding new ACLs in the cluster. Principals must possess Cluster:Alter permissions to call CreateAclsRequest, or be superuser. Unauthorized requests will receive a ClusterAuthorizationException.

```
CreateAclsRequest (Version: 0) => [creation]
  creation => resource_type resource_name principal host operation permission_type
  resource_type => INT8
  resource_name => STRING
  principal => STRING
  host => STRING
  operation => INT8
  permission_type => INT8
```

CreateAclsRequest contains a list of ACLs to add. It can be sent to any broker. The request is not transactional: if one addition fails, the others may proceed. Errors are reported independently for each addition.

```
CreateAclsResponse (Version: 0) => [creation_response]
  creation_response => error_code error_message
  error_code => INT16
  error_message => NULLABLE_STRING
```

There will be a creation_response for each creation in the CreateAclsRequest. The responses will appear in the same order which the requests appeared. For creations which completed successfully, error_code will be 0 and error_message will be null. If there was an error, the error_code will be non-zero and the error_message will give a detailed error message describing why the creation could not be performed.

Because the arguments to CreateAclsRequest are concrete ACLs and not filters, they should not contain ANY or null fields. They also should not contain UNKNOWN fields. Resource names cannot be empty. When the resource type is CLUSTER, the name must be "kafka-cluster".

DeleteAclsRequest and DeleteAclsResponse

DeleteAclsRequest handles removing ACLs. Principals must possess Cluster:Alter permissions to call DeleteAclsRequest, or be superuser. Unauthorized requests will receive a ClusterAuthorizationException.

```
DeleteAclsRequest (Version: 0) => [filter]
  filter => resource_type resource_name principal host operation permission_type
  resource_type => INT8
  resource_name => NULLABLE_STRING
  principal => NULLABLE_STRING
  host => NULLABLE_STRING
  operation => INT8
  permission_type => INT8
```

DeleteAclsRequest contains a list of filters. It will attempt to remove all the ACLs which match each filter.

Just like CreateAclsRequest, DeleteAclsRequest can be sent to any broker. The request is not transactional: if one addition fails, the others may proceed. Results are reported independently for each removal.

```
DeleteAclsResponse (Version: 0) => [filter_response]
  filter_response => error_code error_message [match]
  error_code => INT16
  error_message => NULLABLE_STRING
  match => error_code error_message resource_type resource_name principal host operation permission_type
    error_code => INT16
    error_message => NULLABLE_STRING
    resource_type => INT8
    resource_name => STRING
    principal => STRING
    host => STRING
    operation => INT8
    permission_type => INT8
```

Filter responses will be appear in the same order which the filters appeared. If the filter_response has a non-zero error_code, that means that the filter could not be applied by the server, and the match array will be empty. Otherwise, the match array contains a list of all the ACLs that matched the filter. Each match will have a non-zero error code and error message if it could not be removed. When a filters fails to match an ACLs, it is not an error. This will simply result in getting back a filter_response with an empty match list.

The arguments to DeleteAclsRequest should not contain UNKNOWN fields. The UNKNOWN enum type is intended to represent a server response that can't be fully understood by the client, not a request which the client sends. However, in a filter, ANY or null fields will match UNKNOWN fields. So it is possible to construct a filter that will delete all ACLs attached to a particular topic, even when some of those ACLs contain UNKNOWNs.

If the client is newer than the broker, some of the fields may show up as UNKNOWN on the broker side. In this case, the filter will get an UnsupportedVersionException and the filter will not be applied.

New AdminClient APIs

ResourceType, AclOperation, AclPermissionType

The AdminClient needs some classes to represent the concepts of resource types, ACL operations, and permission types.

```
enum AclResourceType { ... }


enum AclOperation { ... }


enum AclPermissionType { ... }
```

The enumerators have the values described in the "New Network Requests" section, as well as functions to translate between INT8 and the enum types.

AclResource Class

The AclResource class represents a resource that an ACL can be attached to.

```
class AclResource {
  String name;
  AclResourceType resourceType;
}
```

AclData Class

The AclResource class represents data about an ACL itself.

```
class AclData {
  String principal;
  String host;
  AclOperation operation;
  AclPermissionType permissionType;
}
```

AclFixture Class

The AclFixture class represents an ACL and the resource it is attached to.

```
class AclFixture {
  AclResource resource;
  AclData data;
}
```

AdminClient#describeAcls

The listAcls API surfaces ListAclsRequest.

```
DescribeAclsResult AdminClient#describeAcls(AclFixture filter, DescribeAclsOptions options);

public DescribeAclsOptions
  DescribeAclsOptions setTimeout(Integer timeout);
}
```

The "filter" object is a filter which will be used to select which ACLs are reported. Fields which are null or ANY match anything. It is an error to specify fields as UNKNOWN.

The DescribeAclsResult object contains a KafkaFuture with the ACL Descriptions.

```
public DescribeAclsResult {
  public KafkaFuture<List<AclFixture>> all();
}
```

AdminClient#CreateAcls

The CreateAcls API surfaces CreateAclsRequest.

```
CreateAclsResult AdminClient#createAcls(Collection<AclDescription> acls, CreateAclsOptions options);

public CreateAclsOptions
  CreateAclsOptions setTimeout(Integer timeout);
}

public CreateAclsResult {
  public KafkaFuture<Void> all();
  public Map<AclDescription, KafkaFuture<Void>> results();
}
```

AdminClient#deleteAcls

The deleteAcls API surfaces DeleteAclsRequest.

```
DeleteAclsResult AdminClient#deleteAcls(Collection<AclFilter> filters, DeleteAclsOptions options);

public DeleteAclsOptions
  DeleteAclsOptions setTimeout(Integer timeout);
}

public DeleteAclsResult {
  public KafkaFuture<Map<AclFilter, List<AclDescription>> all();
  public Map<AclFilter, List<KafkaFuture<AclDescription>>> results();
}
```

Note that removing a topic does not remove the associated ACLs, nor does removing ACLs remove the associated topic.

New Exceptions

SecurityDisabledException

If no authorizer is configured, and the user attempts to list, add, or remove ACLs, SecurityDisabledException will be thrown. Its error code will be 53.

Compatibility Plan

Since there are no existing ACL APIs and requests, backwards compatibility is not an issue. However, we still need to think about forwards compatibility. The version of the AdminClient that we release in 0.11 should be able to interact with future versions of the broker.

If we later add new resource types, operation types, and so forth, we would like to be able to interact with them with the old AdminClient. This is why the AclResourceType, AclOperation, and AclPermissionType enums have UNKNOWN values. If we get an INT8 which we don't know the name for, it gets mapped to an UNKNOWN object.

What if we later add more dimensions to the 4-tuple that describes ACLs, or the 2-tuple that describes resources? CreateAcls will continue to work, although the entries it creates will always get the default value for the new dimension. DescribeAcls and DeleteAcls will also continue to work. The filters created by older clients will always have an implicit "any" entry for the new dimension. This allows the old AdminClient to continue to be able to function in the new environment.

Rejected Alternatives

Combined CreateAcls and DeleteAcls API (AlterAcls)

We could have combined CreateAcls and DeleteAcls into a single AlterAcls RPC. However, this would have been a bad idea for a few reasons:

- The name "AlterAcls" suggests that ACLs are being altered. However, in fact ACLs are only being added or removed, but not altered.
- It's unclear what order the add and remove operations happen in.
- It is unclear whether a remove operation can remove something added in the same AlterAcls request.
- If add and remove operations are reordered, a security hole could be created when brokers are configured with default-allow behavior. Deleting a restrictive ACL on a secure topic before adding a new restrictive ACL on that topic creates a window of vulnerability.
- CreateAcls and DeleteAcls is similar to the existing AddTopics and RemoveTopics APIs.

Future Work

Once AdminClient supports ACL operations, we can transition the command-line utilities to using it, instead of contacting ZooKeeper directly