KIP-290: Support for Prefixed ACLs

Status

Current state: *"Accepted"*

Discussion thread: *here*

JIRA: *here*

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

Motivation

Kafka authorizes access to resources like topics, consumer groups etc. by way of ACLs. The current supported semantic of resource name in ACL definition is either full resource name or special wildcard **'*'**, which matches everything.

Kafka should support a way of defining bulk ACLs instead of specifying individual ACLs.
Example use cases:

- Principal “user2” has access to all topics that start with “com.company.product1.”.
- Principal “user1” has access to all consumer groups that start with “com.company.client1.”.

Support for adding ACLs to such 'prefixed resource patterns' will greatly simplify ACL operational story in a multi-tenant environment.

Public Interfaces

Summary of changes

- Create new `ResourcePattern` class in `o.a.k.common.resource`. This will be used to represent the resource pattern that ACLs can be attached to. This will be used within the AdminClient implementation and server side code for *create* requests and in *list* and *remove* responses. This will replace the use of the current Resource class.
- Create new `ResourcePatternFilter` class in `o.a.k.common.resource`. This will be used within the AdminClient implementation and server side code for *list* and *remove* requests. This will replace the use of the current ResourceFilter.
- Create new `PatternType` enumeration in `o.a.k.common.resource`. This be used in `ResourcePattern` and `ResourcePatternFilter` to define the type of pattern they represent. With values of:
  - LITERAL: meaning the pattern's name is treated literally. The pattern will only match a resource with the same name. The exception to this is a pattern with the wildcard '*' name, which for backwards compatibility reasons must also be of type literal, and which matches a resource of any name.
  - PREFIXED: meaning the pattern's name is treated as a prefix. The pattern will match any resource with a name that starts with the patterns name.
  - ANY: (For use in filters only) meaning the filter will ignore the pattern's type.
  - MATCH: (For use in filters only) meaning the filter will match any pattern that would match the filters name, e.g. given a filter with name 'payments.received', it would match a literal pattern with the exact same name or the wildcard name and would match a prefixed pattern with a name such as 'payments.'. This value cane be used to select all resource patterns that match a specific resource.
- Update the SimpleAclAuthorizer to support prefixed resource patterns
  - The `kafka.auth.Resource` class will have a new `PatternType` field added, but the class will reject `PatternType.ANY` or `PatternType.MATCH`.
  - The contract of existing CRUD operations on the SimpleAclAuthorizer will remain the same: they will only return ACLs for the resources that exact matches the ones passed in.
  - The authorize(...) method, which currently calls getAcls(resource) and getAcls('*') to get all the matching ACLs today, will instead now look for all matching literal and prefixed ACLs.
  - To avoid resource pattern name clashes in ZK, non-literal ACLs will be stored under a new ZK path: '/kafka-acl-extended/<pattern-type>' and change events to such ACLs will be stored under: '/kafka-acl-extended-changes'.
  - ACL change events stored under the new '/kafka-acl-extended-changes' path in ZK will have a JSON value value.

    - ACL change events stored under the existing '/kafka-acl-changes' path in ZK will continue to use a colon separated value.
- Change the command line tool class AclCommand.scala, (the code behind the `kafka-acls.sh` script).
    - Expose a '--resource-pattern-type' switch, which can be set to literal, prefixed, any or match, e.g.
      `bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 --add --allow-principal User:Bob  --operation Read --group my-app- --resource-pattern-type prefixed`
    - The pattern-type will default to 'literal', meaning the commands will return only the ACLs they previously returned, even if prefixed ACLs exists for the resource.
    - Users can now set the flag to 'match' to retrieve/delete *all* ACLs affecting the supplied ACLs resource(s).
- Changes to the admin client to support prefixed ACLs.
    - New schema version for `CreateAclsRequest`, `DescribeAclsRequest`, `DeleteAclsRequest`, and associated responses, which will have a new byte field containing the pattern type.
    - The `CreateAclRequest` will only accept prefixed or literal patterns.
    - The DescribeAclsRequest and DeleteAclRequest will accept all `PattternTypes`. The use of `LITERAL` and `PREFIXED` will mean only ACLs on those exact resource patterns will be affected. The use of `ANY` will be synonymous with issuing separate commands for both `LITERAL` and `PREFIXED`. The use of `MATCH` will mean the command performs pattern matching, returning / removing all ACLs that affect the supplied resource, including any wildcard patterns.

AdminClient changes

The examples below should help demonstrate the purposed functionality:

Exissting functionality

Each call retrieves ACLs stored in one path in ZK:
`adminClient.describeAcls(... TOPIC, "foobar" ...)` -> would return only ACLs from '/kafka-acl/Topic/foobar' path
`adminClient.describeAcls(... TOPIC, "*" ...)` -> would return only ACLs from '/kafka-acl/Topic/*' path

New functionality

Using legacy constructors

Legacy constructors default resourceNameType to Literal and maintains existing contract:
`adminClient.describeAcls(... TOPIC, "foobar" ...)` -> still returns only ACLs from '/kafka-acl/Topic/foobar' path
`adminClient.describeAcls(... TOPIC, "*" ...)` -> still returns only ACLs from '/kafka-acl/Topic/*' path

Using constructors with explicit resourceNameType

The user needs to *know*, up front, which prefixed resource paths exist to be able to query them:
`adminClient.describeAcls(... TOPIC, "foobar", LITERAL ...)`-> will return only ACLs from '/kafka-acl/Topic/foobar' path
`adminClient.describeAcls(... TOPIC, "*", LITERAL ...)` -> will return only ACLs from '/kafka-acl/Topic/*' path
`adminClient.describeAcls(... TOPIC, "foo", PREFIXED ...)` -> will return only ACLs from '/kafka-acl-extended/prefixed/Topic/foo' path

Using 'ANY' resource name type

`adminClient.describeAcls(... TOPIC, "foo", ANY ...)` -> will return ACLs from both '/kafka-acl/Topic/foobar' and '/kafka-acl-extended /prefixed/Topic/foo' path

Using 'MATCH' resource name type

This will perform pattern matching to allow user to *discover* all the ACLs affecting a specific resource:
`adminClient.describeAcls(... TOPIC, "foobar", MATCH ...)` -> will return all ACLs affecting topic "foobar", including any prefixed and wildcard ACLs.

The return value from `describeAcls` will contain the pattern type for each ACL, so the user can determine if it is literal or prefixed.

Acls on prefixed resource paths are *never* returned to earlier clients. Nor can older clients delete ACLs on prefixed resource paths.

Proposed Changes

Solution
The proposal is to enhance the `SimpleAclAuthorizer`, `AdminClient` and `AclCommand` classes to support prefixed ACLs.
This means that it will be possible to create ACLs of type: User:clientA has READ access on any topic whose name starts with the prefix 'orgA', i.e clientA has READ access to all topics that start with `orgA`.
Currently ACLs are stored against a `Resource` in ZK, i.e. a resource-type and name. This is insufficient to represent prefixes.The solution will introduce the concept of a `ResourcePattern` and ACLs in ZK will be stored against such patterns.

Storage model
Currently, ACLs are stored on ZK under path /kafka-acl/<resource-type>/<resource-name>.

For example:
ACLs for topic '*topicName'* will be stored under '/kafka-acl/Topic/topicName'.
ACLs for consumer group '*group:Id'* will be stored under '/kafka-acl/Group/group:Id'.
ACLs for the wildcard topic '*' will be stored under '/kafka-acl/Topic/*'.

As some resource types, e.g. consumer groups, have freeform topic names it is not possible to store prefixed resource patterns under the same ZK root, as there is no viable encoding of the pattern type into the path that does not have the potential of name clashes with existing or future literal patterns. e.g. consider encoding prefixed patterns using paths such as '/kafka-acl/Group/prefixed/my-group' or '/kafka-acl/Group/prefixed:my-group' - both can clash with potential existing paths: the first would match a group named 'prefixed/my-group' and the second one named 'prefixed:my-group', both of which are valid group names.

To work around the free-form nature of some resource types, ACLs for the new prefixed resource patterns will be stored under a second root in ZK: '/kafka-acl-extended/prefixed'. Changes will first be stored under '/kafka-acl-extended-changes'. We will also take the opportunity to store change events as JSON under the new path, allowing for easier enhancements in the future.

Literal resources, including the wildcard resource '*', will continue to be stored in their original location.

Access will be allowed if there is at least one ALLOW matching acl and no DENY matching ACL (current behaviour is maintained). Note that the length of the prefix doesn't play any role here.

Extended ACL storage

$ get /kafka-extended-acl/prefixed/Topic/orgName
{"version":1,"acls":[{"principal":"User:clientA","permissionType":"Allow","operation":"Read","host":""}]}*

Extended ACL change event storage

$ get /kafka-acl-extended-changes/acl_changes_0000
{"version":1, "resourceType":"Topic", "name":"orgName", "patternType":"PREFIXED"}

Compatibility, Deprecation, and Migration Plan

On downgrade, any prefixed ACLs will be ignored because they are in separate path. This means that any prefixed ACLs will be treated as if they were never added. This is fine for ALLOW ACLs, but might have security implications if DENY ACLs are ignored.

Constructors for Resource and ResourceFilter that don't take a ResourceNameType will be deprecated in favour of those that do.

Care has been taken to ensure legacy clients can neither add, list or remove prefixed ACLs. Clients wishing to use prefixed ACLs will need to upgrade their (admin)clients.

Rejected Alternatives

- Use escaping to identify prefixed vs literals (won't work on ZK). We decided to use a separate path for prefixed ACLs