KIP-801: Implement an Authorizer that stores metadata in __cluster_metadata

Status

Current state: *Approved*

Discussion thread: DISCUSS DISCUSS+VOTE

JIRA: KAFKA-13646

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

Motivation

Currently, when using KRaft mode, users still have to have an Apache ZooKeeper instance if they want to use *AclAuthorizer*. We should have a built-in Authorizer for KRaft mode that does not depend on ZooKeeper. This KIP introduces such an authorizer, called *StandardAuthorizer*.

Design

Overview

This KIP creates *org.apache.kafka.metadata.authorizer.StandardAuthorizer,* a new Authorizer class which stores its ACLs in the *__cluster_metadata* topic. It is used by default in KRaft clusters, unless the administrator configures a different Authorizer by setting *authorizer.class.name.* Note that the default for ZooKeeper-based clusters remains *AclAuthorizer*.

In general, *StandardAuthorizer* acts as a drop-in replacement for *AclAuthorizer*. It implements all the same behaviors. It also supports the same static configuration keys as *AclAuthorizer*, including *super.users* and *allow.everyone.if.no.acl.found***.**

Consistency

StandardAuthorizer stores ACLs in the *__cluster_metadata* topic. As described in KIP-500 and KIP-631, brokers and standby controllers continuously read this log, up to its last stable offset. This means that at any given time, they will have an Authorizer state corresponding to some point on a single timeline. This is similar to the consistency guarantees we provide today with the ZooKeeper-based *AclAuthorizer*.

Just as with all other metadata state, the authorization state on the active controller may be slightly ahead of the rest of the cluster. If the active controller succeeds in persisting this new state to the log (by advancing the last stable offset) all other nodes will pick it up. Otherwise, if the active controller fails, a new controller will take over and resume from the last stable offset. See KIP-631 for more information.

Ordering

ACL records must be applied in order, even if they appear in the same batch of records. This is important because *StandardAuthorizer*, like other Authorizers, is multi-threaded and will continue to authorize new operations even while record changes are applied.

For example, if we have two ACL records like this:

1. Deny user bob access to topic *foo*
2. Allow user bob access to all topics

We know that when both an ALLOW and a DENY ACL apply to the same access attempt, access must be denied. Therefore, the net effect of applying #1 and then #2 is to give bob access to all topics except for foo. However, if we apply record #2 before record #1, this will result in a brief window of time during which user bob incorrectly has access to topic foo. We must avoid such situations by always applying ACL records in the order they appear in the __ *cluster_metadata* log.

Initialization

In general, Authorizer implementations are expected to block for a certain amount of time when they start up, while they are loading their ACLs. It would be inappropriate to start authorizing requests immediately, even before this ACL data was present. To see why, consider the situation where *allow.everyone.if.no.acl.found* is set to true. When this configuration is set, and we have not yet loaded any ACLs, anyone could access the system. Clearly, having a brief period during startup when anyone can access the system is not acceptable.

Of course, in a system based on streaming metadata changes, there is no "final change" to apply, no terminal state. However, by waiting to start the authorizer until we have replayed ACL records up to the high water mark, we can at least ensure that we don't expose an authorizer state from the past to users during broker and controller startup.

Similarly, when a node has fallen behind and must apply a snapshot, that snapshot should be applied atomically.

Bootstrapping

When we have just created a new cluster, there are no ACL records present yet. This presents a problem: the nodes must communicate with the controller quorum, but there are no ACLs which could allow them to do that. We can solve this problem in one of two ways: by setting *allow.everyone.if.no.acl.found* t o true, or by putting the user(s) for the brokers and controllers into *super.users*. Since most system administrators prefer to operate in a "deny by default" regime, we expect the second solution to be used most often.

In the future, we plan on making more sophisticated KRaft bootstrapping options available – for example, commands that could set up the local metadata log on each node during installation. This will also be useful for things like dynamically configured SCRAM setups, and so forth. However, the simple solution described above should suffice for now.

Early Start Listeners

We know that any user that appears in *super.users* will always be authorized, even if the *StandardAuthorizer* loading process is not yet complete. This lets us implement a useful optimization: if a listener appears in *early.start.listeners*, we will immediately start that listener, even before *StandardAuthorizer* has completed loading up to the __cluster_metadata high water mark. However, until loading has fully completed, we will only allow super users to send requests to this endpoint. All other users will get AUTHORIZER_NOT_READY.

Public Interfaces

Records

AccessControlEntryRecord

This record stores information about access controls that have been put in place in the cluster.

This is basically the same information that is stored in AclBinding, except that it contains a unique ID (UUID) to identify the ACL.

When an AccessControlEntryRecord appears in the log, it signals that a new ACL has been created.

When an AccessControlEntryRecord appears in a snapshot, it describes an ACL that exists in the cluster. The ACLs in the snapshot do not appear in any deterministic order.

```
{
  "apiKey": ...,
  "type": "metadata",
  "name": "AccessControlEntryRecord",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "Id", "type": "uuid", "versions": "0+",
      "about": "The unique ID of this ACL." },
    { "name": "ResourceType", "type": "int8", "versions": "0+",
      "about": "The resource type." },
    { "name": "ResourceName", "type": "string", "versions": "0+",
      "about": "The resource name." },
    { "name": "PatternType", "type": "int8", "versions": "0+",
      "about": "The resource name pattern type." },
    { "name": "Principal", "type": "string", "versions": "0+",
      "about": "The principal name." },
    { "name": "Host", "type": "string", "versions": "0+",
      "about": "The host name." },
    { "name": "Operation", "type": "int8", "versions": "0+",
      "about": "The AclOperation." },
    { "name": "PermissionType", "type": "int8", "versions": "0+",
      "about": "The AclPermissionType." }
  ]
}
```

RemoveAccessControlRecord

When it appears in the metadata log, this record indicates that an ACL has been deleted. This record does not appear in the metadata snapshot.

```
{
  "apiKey": ...,
  "type": "metadata",
  "name": "RemoveAccessControlEntryRecord",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "Id", "type": "uuid", "versions": "0+",
      "about": "The ID of the AccessControlEntry to remove." }
  ]
}
```

Configuration Keys

Key Name	Description	Valid Values	Default Value
early.start.listeners	A list of listeners which we want to start as early as possible. This is useful in cases where the startup process requires some listeners to be open before other listeners can be brought up. In general, a listener should not appear in this list if it accepts external traffic.	A comma-separated list of listener names	The controller listener, if one is present (i.e., if we are in KRaft mode).
super.users	Just as in AclAuthorizer, this is a semi-colon separated list of users that will be treated as super users.	A comma-separated list of user names	Empty
allow.everyone.if.no.acl.found	Just as in AclAuthorizer, if this is set to true, in the case when no acls are found for a resource, the authorizer allows access to everyone.	true | false	false

EnvelopeRequest

We will bump the version of EnvelopeRequest to reflect the fact that it can now return a new error code: AUTHORIZER_NOT_READY. This error code will only ever be returned from early start endpoints. It indicates that the operation could not be performed because the Authorizer has not fully initialized yet, as described in the previous section about "early start listeners."

As described in KIP-590, brokers use EnvelopeRequest to forward user requests to KRaft controllers. When the broker gets back an AUTHORIZER_NOT_READY error for a forwarded request, it should wait for a while and then try to forward the request again. This is basically the same behavior as the what the broker does when there is a network error when forwarding a request. At some point, of course, the request will time out.

If the version of EnvelopeRequest is too old, then the controller will return UNKNOWN_SERVER_EXCEPTION instead.

Since the controller uses ApiVersions to determine what RPC versions to use, rather than consulting the IBP, we do not need to bump the IBP to make this change.

Metadata Shell

The metadata shell will support examining KRaft ACLs. Each ACL will appear in /acl/id/<uuid> in its JSON form.

New Metrics

In order to improve manageability, we will add a new metric, AclCount.

Attribute Name	Value	Notes
kafka.server:type=Authorizer,name=AclCount	Current number of ACLs	For combined nodes, this is the count from the controller authorizer, not the broker authorizer.

New Authorizer Function

In order to support the AclCount metric, we will extend the Authorizer API with a new aclCount function.

```
int aclCount()
```

In order to preserve compatibility, this function will default to returning -1, so that existing Authorizer subclasses will continue to work. Authorizers that expose this metric should override this function with the correct value.

Compatibility, Deprecation, and Migration Plan

ZooKeeper-Based Clusters

This KIP has no impact on ZooKeeper-based clusters. The changes only apply to KRaft-based clusters.

If a user attempts to configure StandardAuthorizer when running in ZK mode, a fatal exception will be thrown indicating the problem.

New Default

For KRaft-based clusters, users that currently use AclAuthorizer can continue to do so. However, they will need to configure AclAuthorizer explicitly rather than getting it as a default.

Migration

If a user transitions to a different authorizer, the previous ACLs stored by StandardAuthorizer still remains stored in __cluster_metadata. They will not be deleted.

Note that we do not currently have any way of loading ACLs from ZooKeeper into this authorizer. However, we will design a mechanism for doing this in a future KIP where we define how to do migration from a ZooKeeper-based Kafka cluster to a KRaft-based Kafka cluster.

Rejected Alternatives

Leave AclAuthorizer as the Default

We could leave AclAuthorizer as the default for KRaft clusters. However, this would be highly confusing to new users, since it would introduce a ZooKeeper dependency when running in KRaft mode. Since KRaft is in Preview mode, we feel that now is the best time to change the default Authorizer for KRaft mode.