Authorization and ACLs¶

Overview¶

Kafka ships with a pluggable Authorizer and an out-of-box authorizer implementation that uses ZooKeeper to store all the ACLs. It is important to set ACLs because otherwise access to resources is limited to super users when an authorizer is configured. The default behavior is that if a resource has no associated ACLs, then no one is allowed to access the resource, except super users.

Broker Configuration¶

Authorizer¶

To enable ACLs, you must configure an authorizer. Kafka provides a simple authorizer implementation, and to use it, you can add the following to `server.properties`:

```
authorizer.class.name=kafka.security.auth.SimpleAclAuthorizer
```

Copy

Super Users¶

By default, if no resource patterns match a specific resource, then the resource has no associated ACLs, and therefore no one other than super users is allowed to access the resource. If you want to change that behavior, you can include the following in `server.properties`:

```
allow.everyone.if.no.acl.found=true
```

Copy

You can also add super users in `server.properties` like the following (note that the delimiter is semicolon since SSL user names may contain comma):

```
super.users=User:Bob;User:Alice
```

Copy

User Names¶

By default, the SSL user name will be of the form `CN=writeuser,OU=Unknown,O=Unknown,L=Unknown,ST=Unknown,C=Unknown`. One can change that by setting a customized PrincipalBuilder in `server.properties` like the following:

```
principal.builder.class=CustomizedPrincipalBuilderClass
```

Copy

By default, the SASL user name will be the primary part of the Kerberos principal. One can change that by setting `sasl.kerberos.principal.to.local.rules` to a customized rule in `server.properties`.

In the event that SSL is enabled but client authentication is not configured, clients will connect anonymously via the `SSL` port and will appear to the server with the user name `ANONYMOUS`. Such a configuration provides encryption and server authentication, but clients will connect anonymously. The other case in which the server will see the `ANONYMOUS` user is if the `PLAINTEXT` security protocol is being used. By giving read/write permission to the `ANONYMOUS` user, you are allowing anyone to access the brokers without authentication. As such, you typically do not want to give access to `ANONYMOUS` users unless the intention is to give everyone the permission.

Using ACLs¶

The following examples use `bin/kafka-acls` (the Kafka Authorizer CLI) to add, remove or list ACLs. For detailed information on the supported options, run `bin/kafka-acls --help`. Note that ACLs are stored in ZooKeeper and they are propagated to the brokers asynchronously so there may be a delay before the change takes effect even after the command returns.

ACL Format¶

Kafka ACLs are defined in the general format of “Principal P is [Allowed/Denied] Operation O From Host H On Resource R matching ResourcePattern RP”. The following table describes the relationship between operations, resources and APIs:

Operation	**Resource**	**API**
ALTER	Cluster	AlterReplicaLogDirs
ALTER	Cluster	CreateAcls
ALTER	Cluster	DeleteAcls
ALTER	Topic	CreatePartitions
ALTER_CONFIGS	Cluster	AlterConfigs
ALTER_CONFIGS	Topic	AlterConfigs
CLUSTER_ACTION	Cluster	Fetch (for replication only)
CLUSTER_ACTION	Cluster	LeaderAndIsr
CLUSTER_ACTION	Cluster	OffsetForLeaderEpoch
CLUSTER_ACTION	Cluster	StopReplica
CLUSTER_ACTION	Cluster	UpdateMetadata
CLUSTER_ACTION	Cluster	ControlledShutdown
CLUSTER_ACTION	Cluster	WriteTxnMarkers
CREATE	Cluster	CreateTopics
CREATE	Cluster	Metadata if `auto.create.topics.enable`
CREATE	Topic	Metadata if `auto.create.topics.enable`
CREATE	Topic	CreateTopics
DELETE	Group	DeleteGroups
DELETE	Topic	DeleteRecords
DELETE	Topic	DeleteTopics
DESCRIBE	Cluster	DescribeAcls
DESCRIBE	Cluster	DescribeLogDirs
DESCRIBE	Cluster	ListGroups
DESCRIBE	DelegationToken	DescribeTokens
DESCRIBE	Group	DescribeGroup
DESCRIBE	Group	FindCoordinator
DESCRIBE	Group	ListGroups
DESCRIBE	Topic	ListOffsets
DESCRIBE	Topic	Metadata
DESCRIBE	Topic	OffsetFetch
DESCRIBE	Topic	OffsetForLeaderEpoch
DESCRIBE	TxnId	FindCoordinator
DESCRIBE_CONFIGS	Cluster	DescribeConfigs

Operation	Resource	API
DESCRIBE_CONFIGS	Topic	DescribeConfigs
IDEMPOTENT_WRITE	Cluster	InitProducerId
IDEMPOTENT_WRITE	Cluster	Produce
READ	Group	AddOffsetsToTxn
READ	Group	Heartbeat
READ	Group	JoinGroup
READ	Group	LeaveGroup
READ	Group	OffsetCommit
READ	Group	OffsetFetch
READ	Group	SyncGroup
READ	Group	TxnOffsetCommit
READ	Topic	Fetch
READ	Topic	OffsetCommit
READ	Topic	TxnOffsetCommit
WRITE	Topic	Produce
WRITE	TxnId	Produce
WRITE	Topic	AddPartitionsToTxn
WRITE	TxnId	AddPartitionsToTxn
WRITE	TxnId	AddOffsetsToTxn
WRITE	TxnId	EndTxn
WRITE	TxnId	InitProducerId
WRITE	TxnId	TxnOffsetCommit

The operations in this table are both for clients (producers, consumers, admin) and inter-broker operations of a cluster. In a secure cluster, both client requests and inter-broker operations require authorization. The inter-broker operations are split into two classes: cluster and topic. Cluster operations refer to operations necessary for the management of the cluster, like updating broker and partition metadata, changing the leader and the set of in-sync replicas of a partition, and triggering a controlled shutdown.

Because of the way replication of topic partitions work internally, it is also important to grant topic access to brokers. Brokers replicating a partition must be authorized for both READ and DESCRIBE on that topic. DESCRIBE is granted by default with the READ authorization.

You can use these methods to automatically grant topic access to servers:

Make the server principal a super user. By configuring the cluster this way, servers can automatically access all resources, including the cluster resource.

Use the wildcard for topics so that you only have to set it once. You must set an ACL for the cluster resource separately. For example:

```
kafka-acls --authorizer-properties zookeeper.connect=localhost:2181 --add \
--allow-principal User:Alice --operation All --topic '*' --cluster
```

Copy

Producers and consumers need to be authorized to perform operations on topics, but they should be configured with different principals compared to the servers. The main operations that producers require authorization to execute are WRITE and READ. Admin users can execute command line tools and require authorization. Operations that an admin user might need authorization for are DELETE, CREATE, and ALTER. You can use wildcards for producers and consumers so that you only have to set it once.

Wildcards are any resource, including groups.

You can give topic and group wildcard access to users who have permission to access all topics and groups (e.g admin users). If you use this method, you don’t have to create a separate rule for each topic and group for the user. For example, you can use this to give wildcard access to Alice:

```
kafka-acls --authorizer-properties zookeeper.connect=localhost:2181 --add --allow-principal \
User:Alice --operation All --topic '*' --group '*'
```

Copy

Common cases:¶

create a topic, the principal of the client will require the CREATE and DESCRIBE operations on the `topic` resource (via the *Metadata* API with `auto.create.topics.enable`).
produce to a topic, the principal of the producer will require the WRITE operation on the `topic` resource.
consume from a topic, the principal of the consumer will require the READ operation on the `topic` and `group` resources.

Note that to be able to create, produce, and consume, the servers need to be configured with the appropriate ACLs. The servers need authorization to update metadata (CLUSTER_ACTION) and to read from a topic (READ) for replication purposes.

Adding ACLs¶

Suppose you want to add an ACL “Principals User:Bob and User:Alice are allowed to perform Operation Read and Write on Topic test-topic from IP 198.51.100.0 and IP 198.51.100.1”. You can do that by executing the following:

```
bin/kafka-acls --authorizer-properties zookeeper.connect=localhost:2181 --add \
 --allow-principal User:Bob --allow-principal User:Alice \
 --allow-host 198.51.100.0 --allow-host 198.51.100.1 --operation Read --operation Write --topic test-topic
```

Copy

By default all principals that don’t have an explicit ACL allowing an operation to access a resource are denied. In rare cases where an ACL that allows access to all but some principal is desired, you can use the `--deny-principal` and `--deny-host` options. For example, use the following command to allow all users to Read from `test-topic` but only deny `User:BadBob` from IP 198.51.100.3:

```
bin/kafka-acls --authorizer-properties zookeeper.connect=localhost:2181 --add \
 --allow-principal User:* --allow-host * --deny-principal User:BadBob --deny-host 198.51.100.3 \
 --operation Read --topic test-topic
```

Copy

Note that `--allow-host` and `deny-host` only support IP addresses (hostnames are not supported).

The examples above add ACLs to a topic by specifying `--topic [topic-name]` as the resource pattern option. Similarly, one can add ACLs to a cluster by specifying `--cluster` and to a group by specifying `--group [group-name]`. In the event that you want to grant permission to all groups, you may do so by specifying `--group=*` as shown in the following command:

```
bin/kafka-acls --authorizer kafka.security.auth.SimpleAclAuthorizer \
  --authorizer-properties zookeeper.connect=localhost:2181 --add \
  --allow-principal User:* --operation read --topic test --group=*
```

Copy

You can add acls on prefixed resource patterns. For example, you can add an acl for user Jane to produce to any topic whose name starts with `Test-`. You can do that by executing the CLI with following options:

```
bin/kafka-acls --authorizer-properties zookeeper.connect=localhost:2181 --add --allow-principal \
  User:Jane --producer --topic Test- --resource-pattern-type prefixed
```

Copy

Note that `--resource-pattern-type` defaults to `literal`, which only affects resources with the exact same name or, in the case of the wildcard resource name ‘*’, a resource with any name.

Removing ACLs¶

Removing ACLs is similar, but the `--remove` option should be specified instead of `--add`. To remove the ACLs added in the first example above you can execute the following:

```
bin/kafka-acls --authorizer-properties zookeeper.connect=localhost:2181 --remove \
 --allow-principal User:Bob --allow-principal User:Alice \
 --allow-host 198.51.100.0 --allow-host 198.51.100.1 \
 --operation Read --operation Write --topic test-topic
```

Copy

If you want to remove the acl added to the prefixed resource pattern in the example we can execute the CLI with following options:

```
bin/kafka-acls --authorizer-properties zookeeper.connect=localhost:2181 --remove \
  --allow-principal User:Jane --producer --topic Test- --resource-pattern-type Prefixed
```

Copy

Listing ACLs¶

You can list the ACLs for a given resource by specifying the `--list` option and the resource. For example, to list all ACLs for `test-topic` you can execute the following:

```
bin/kafka-acls --authorizer-properties zookeeper.connect=localhost:2181 \
 --list --topic test-topic
```

Copy

However, this will only return the acls that have been added to this exact resource pattern. Other acls can exist that affect access to the topic, e.g. any acls on the topic wildcard '*', or any acls on prefixed resource patterns. Acls on the wildcard resource pattern can be queried explicitly:

```
bin/kafka-acls --authorizer-properties zookeeper.connect=localhost:2181 --list --topic *
```

Copy

It is not necessarily possible to explicitly query for acls on prefixed resource patterns that match Test-topic as the name of such patterns may not be known. We can list all acls affecting Test-topic by using `--resource-pattern-type match`. For example:

```
bin/kafka-acls --authorizer-properties zookeeper.connect=localhost:2181 --list --topic Test-topic --resource-pattern-type match
```

Copy

This will list acls on all matching literal, wildcard and prefixed resource patterns.

Adding or Removing a Principal as Producer or Consumer¶

The most common use cases for ACL management are adding/removing a principal as a producer or consumer. To add `User:Bob` as a producer of `test-topic` you can execute the following:

```
bin/kafka-acls --authorizer-properties zookeeper.connect=localhost:2181 \
 --add --allow-principal User:Bob \
 --producer --topic test-topic
```

Copy

To add Alice as a consumer of `test-topic` with group `Group-1`, you can specify the `--consumer` and `--group` options:

```
bin/kafka-acls --authorizer-properties zookeeper.connect=localhost:2181 \
 --add --allow-principal User:Bob \
 --consumer --topic test-topic --group Group-1
```

Copy

To remove a principal from a producer or consumer role, you can specify the `--remove` option.

Authorization in the REST Proxy and Schema Registry¶

You may use Kafka ACLs to enforce authorization in the REST Proxy and Schema Registry. These require Confluent security plugins (../confluent-security-plugins/index.html#confluentsecurityplugins-introduction).

Debugging¶

It's possible to run with authorizer logs in `DEBUG` mode by making some changes to the `log4j.properties` file. If you're using the default `log4j.properties` file in the Confluent Platform 0.9.0.0 or above, you simply need to change the following line to `DEBUG` mode instead of `WARN`:

```
log4j.logger.kafka.authorizer.logger=WARN, authorizerAppender
```

Copy

The `log4j.properties` file is located in the Kafka config directory at `/etc/kafka/log4j.properties`. In the event that you're using an earlier version of the Confluent Platform, or if you're using your own `log4j.properties` file, you'll need to add the following lines to the config:

```
log4j.appender.authorizerAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appender.authorizerAppender.DatePattern='.'yyyy-MM-dd-HH
log4j.appender.authorizerAppender.File=${kafka.logs.dir}/kafka-authorizer.log
log4j.appender.authorizerAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.authorizerAppender.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.logger.kafka.authorizer.logger=DEBUG, authorizerAppender
log4j.additivity.kafka.authorizer.logger=false
```

Copy

You'll need to restart the broker before it will take effect. This will log every request being authorized and its associated user name. The log is located in `$kafka_logs_dir/kafka-authorizer.log`. The location of the logs depends on the packaging format - `kafka_logs_dir` will be in `/var/log/kafka` in rpm/debian and `$base_dir/logs` in the archive format.
