KIP-608 - Expose Kafka Metrics in Authorizer

Status

Current state: *Accepted*

Discussion thread: *here*

Vote thread: *here*

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

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

Motivation

Current Kafka authorizer plugin cannot access to Kafka broker metrics information. Kafka authorizer plugins need access to runtime broker Metrics instance to add additional metrics. There is disconnection between how to manage Metrics between broker and authorizer plugins in current Kafka implementation. The authorizer plugins as plugins of broker could use same Metrics instance in broker, so authorizer plugins need not manage tasks like creating and configuring Metrics and JmxReporter. With the feature of this KIP, authorizer plugin can use broker's Metrics instance to add additional metrics very easy.

Public Interfaces

Define a new interface Monitorable in package org.apache.kafka.common. The authorizer plugins can implement the interface and broker can pass the instance of Metrics to Authorizer. Other broker'd or client's plugins could potentially implement the interface in the future and get the broker's or client's Metrics instance to add additional metrics from sub-components. Support for other plugins will be addressed later in another KIP if necessary. This KIP will focus on exposing Kafka Metrics in Authorizer only.

Monitorable Interface

```
package org.apache.kafka.common;

import org.apache.kafka.common.metrics.Metrics;

/**
 * Interface for plugins to get Metrics instance
 */
public interface Monitorable {
    /**
     * Get the instance of {@link Metrics}.
     */
    default void monitor(Metrics metrics) {
                return;
        };
}
```

The org.apache.kafka.server.authorizer.Authorizer interface need be updated to extend Monitorable Interface.

Authorizer Interface

```
package org.apache.kafka.server.authorizer.Authorizer
...
import org.apache.kafka.common.Monitorable;
...
public interface Authorizer extends Configurable, Closeable, Monitorable {
...
}
```

The following metrics will be added:

Full Name	Type	Description
kafka.server:type=kafka.security.authorizer.metrics,name=acls-total-count	32-bit gauge	Total acls created in the broker
kafka.server:type=kafka.security.authorizer.metrics,name=authorization-request-rate-per-minute	Rate per minute	Total number of authorization requests per minute
kafka.server:type=kafka.security.authorizer.metrics,name=authorization-allowed-rate-per-minute	Rate per minute	Total number of authorization allowed per minute
kafka.server:type=kafka.security.authorizer.metrics,name=authorization-denied-rate-per-minute	Rate per minute	Total number of authorization denied per minute

Proposed Changes

kafka.server.KafkaServer will be updated to pass instance of Metrics to Authorizer.

kafka.server.KafkaServer

```
/* Get the authorizer and initialize it if one is specified.*/
authorizer = config.authorizer
authorizer.foreach(_.configure(config.originals))
authorizer.foreach(_.monitor(metrics))
...
```

kafka.security.authorizer.AclAuthorizer will be updated to collect and add Authorizer metrics.

kafka.security.authorizer.AclAuthorizer

```
class AclAuthorizer extends Authorizer with Logging {
......
        private var authorizerMetrics: AuthorizerMetrics = _
......
        override def monitor(metrics: Metrics): Unit = {
            authorizerMetrics = new AuthorizerMetrics(metrics)
          }
......

        private def authorizeAction(requestContext: AuthorizableRequestContext, action: Action):
AuthorizationResult = {
                ......
                logAuditMessage(requestContext, action, authorized)
                authorizerMetrics.recordAuthorizerMetrics(authorized)
            if (authorized) AuthorizationResult.ALLOWED else AuthorizationResult.DENIED
        }
...

        class AuthorizerMetrics(metrics: Metrics) {
            val GROUP_NAME = "kafka.security.authorizer.metrics"
            val authorizationAllowedSensor = metrics.sensor("authorizer-authorization-allowed")
            authorizationAllowedSensor.add(metrics.metricName("authorization-allowed-rate-per-minute",
GROUP_NAME,
                      "The number of authoization allowed per hour"), new Rate(TimeUnit.MINUTES, new
WindowedCount()))

            val authorizationDeniedSensor = metrics.sensor("authorizer-authorization-denied")
            authorizationDeniedSensor.add(metrics.metricName("authorization-denied-rate-per-minute", GROUP_NAME,
                      "The number of authoization denied per hour"), new Rate(TimeUnit.MINUTES, new
WindowedCount()))

            val authorizationRequestSensor = metrics.sensor("authorizer-authorization-request")
            authorizationRequestSensor.add(metrics.metricName("authorization-request-rate-per-minute",
GROUP_NAME,
                      "The number of authoization request per hour"), new Rate(TimeUnit.MINUTES, new
WindowedCount()))

            metrics.addMetric(metrics.metricName("acls-total-count", GROUP_NAME, "The number of acls defined"),
(config, now) => aclCache.size)

            def recordAuthorizerMetrics(authorized: Boolean): Unit = {
                      if (authorized) {
                        authorizationAllowedSensor.record()
                      } else {
                        authorizationDeniedSensor.record()
                      }
                      authorizationRequestSensor.record()
            }
          }
}
```

Compatibility, Deprecation, and Migration Plan

Broker will call authorizer's monitor(metrics: Metrics) which has default implementation after authorizer configured. Old version of authorizer plugins doesn't call the monitor method, so it is backward to old version of authorizer plugins.

Test Plan

Test total number of acls created match with the number from metrics

Test authorization request rate per minute

Test authorization allowed rate per minute

Test authorization denied rate per minute

Rejected Alternatives

Authorizer plugins create and configure own Metrics and JmxReport. Using this alternative approach make collecting Kafka metrics in Authorizer plugin separate from broker metrics that cause disconnection. Users need implement and manage own Metrics and JmsReport with extra effort.

Pass the instance of Metrics through AuthorizerServerInfo interface. The instance of Metrics should not be part of Authorizer server information. We use an interface to expose Kafka Metrics so that other broker's or client's plugins could potentially implement the interface.