KIP-567: Kafka Cluster Audit

Status

Current state: *Under Discussion*

Discussion thread: *new discussion* and *old discussion*

JIRA: *KAFKA-9413*

Motivation

Auditing is a reporting functionality to notify other subsystems of the outcome of an authorization. It is used to check the activity of certain entities within a cluster. It is highly demanded in most businesses to have the ability of obtaining audit information in case someone changes cluster configuration (like creation/deletion/modify/description of any topic or ACLs) or even record client events in some environment. Audits need to be structured as the implementors of this auditing would be JVM based applications and a more loose format (such as as emitting JSON as KIP-673) would in fact provide a less safe API.

As a broader requirement reporting can happen to multiple services attached to Kafka each capturing different aspects and not just the outcome of the authorization but the outcome of the whole action. In a simple use-case one might log topic create events with the information whether they were authorized and successful or not. In a more complicated use-case one can report client and topic events into Apache Atlas which can create a dependency graph of these events to visualize the interconnectedness of clients and topics.

In this KIP we try to provide a generic solution that can be applied to a broader interpretation of auditing which can be applied to a variety of use-cases, such as the ones described above in a similar implementation fashion as the Authorizer.

A video recording is available below.

Public Interfaces

Auditor

Developers will be required to implement an interface which gives the extension point to implement auditing and reporting. In terms of form it can make sense to provide a similar interface to the `Authorizer` as they are closely related, they are the flip side of each other. The snippet defining the interface can be found below. Similarly to the `Authorizer` we implement this in Java in the clients module so implementors won't have to depend on the core module and ultimately on Scala.

To describe the interface broadly, it provides the request, its context, the authorized action and resources with the outcome of the authorization and errors if there were any. It also required to be an asynchronous implementation with low latency as it taps into performance-sensitive areas, such as handling produce requests. Resources can be created and destroyed with the `configure()` and `close()` methods. Moreover exactly one audit call will happen when calling a certain API as authorizations will be collected throughout the API and passed to the auditor when all information is available, therefore giving the widest possible context to the implementer.

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements. See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License. You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.apache.kafka.server.auditor;

import org.apache.kafka.common.Configurable;
import org.apache.kafka.common.annotation.InterfaceStability;
import org.apache.kafka.server.authorizer.AuthorizableRequestContext;

/**
 * An auditor class that can be used to hook into the request after its completion and do auditing tasks, such
as
 * logging to a special file or sending information about the request to external systems.
 * Threading model:
 * <ul>
 *     <li>The auditor implementation must be thread-safe.</li>
 *     <li>The auditor implementation is expected to be asynchronous with low latency as it is used in
performance
 *     sensitive areas, such as in handling produce requests.</li>
 *     <li>Any threads or thread pools used for processing remote operations asynchronously can be started
during
 *     start(). These threads must be shutdown during close().</li>
 * </ul>
 */
@InterfaceStability.Evolving
public interface Auditor extends Configurable, AutoCloseable {

    /**
     * Called on request completion before returning the response to the client. It allows auditing multiple
resources
     * in the request, such as multiple topics being created.
     * @param event is the request specific data passed down to the auditor. It may be null if there are no
specific
     *              information is available for the given audited event type.
     * @param requestContext contains metadata to the request.
     */
    void audit(AuditEvent event, AuthorizableRequestContext requestContext);
}
```

AuditInfo

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements. See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License. You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
```

```
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.apache.kafka.server.auditor;

import org.apache.kafka.common.acl.AclOperation;
import org.apache.kafka.common.annotation.InterfaceStability;
import org.apache.kafka.common.resource.ResourcePattern;
import org.apache.kafka.server.authorizer.AuthorizationResult;

/**
 * This class encapsulates the authorization information with the result of the operation. It is used in
specific ways
 * in the {@link AuditEvent} implementations. For instance a {@link org.apache.kafka.server.auditor.events.
TopicEvent}
 * will have an AuditInfo for every topic as each topic is authorized but in case of an
 * {@link org.apache.kafka.server.auditor.events.AclEvent} authorization only happens for the cluster resource,
 * therefore there will be only one instance of this.
 */
@InterfaceStability.Evolving
public class AuditInfo {

    private final ResourcePattern resourcePattern;
    private final AclOperation operation;
    private final AuthorizationResult allowed;
    private final int error;

    public AuditInfo(AclOperation operation, ResourcePattern resourcePattern) {
        this.operation = operation;
        this.resourcePattern = resourcePattern;
        this.allowed = AuthorizationResult.ALLOWED;
        this.error = 0;
    }

    public AuditInfo(AclOperation operation, ResourcePattern resourcePattern, AuthorizationResult allowed, int
error) {
        this.operation = operation;
        this.resourcePattern = resourcePattern;
        this.allowed = allowed;
        this.error = error;
    }

    public AclOperation operation() {
        return operation;
    }

    public ResourcePattern resource() {
        return resourcePattern;
    }

    public AuthorizationResult allowed() {
        return allowed;
    }

    public int errorCode() {
        return error;
    }
}
```

The KIP also introduces a new configuration called `auditors` which is a comma-separated list of `Auditor` implementations. By default it is configured with the `LoggingAuditor` default implementation.

Property settings example

```
auditors=org.apache.kafka.server.auditor.LoggingAuditor,org.whatever.OtherAuditor
```

AuditEvent

This is a marker interface to serve as a base for all specific event class implementations.

AuditEvent

```
package org.apache.kafka.server.auditor;

import org.apache.kafka.common.annotation.InterfaceStability;

@InterfaceStability.Evolving
public interface AuditEvent {
}
```

Specific Event Classes

There will be specific classes defined for each event much like the *Result classes for the AdminClient. Some examples are:

TopicEvent

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements. See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License. You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.apache.kafka.server.auditor.events;

import org.apache.kafka.common.annotation.InterfaceStability;
import org.apache.kafka.server.auditor.AuditEvent;
import org.apache.kafka.server.auditor.AuditInfo;

import java.util.Collections;
import java.util.Map;
import java.util.Objects;

@InterfaceStability.Evolving
public class TopicEvent extends AuditEvent {

    public static class AuditedTopic {
        private final String topicName;
        private final int numPartitions;
        private final int replicationFactor;

        private static final int NO_PARTITION_NUMBER = -1;
        private static final int NO_REPLICATION_FACTOR = -1;

        public AuditedTopic(String topicName) {
```

```
            this.topicName = topicName;
            this.numPartitions = NO_PARTITION_NUMBER;
            this.replicationFactor = NO_REPLICATION_FACTOR;
        }

        public AuditedTopic(String topicName, int numPartitions, int replicationFactor) {
            this.topicName = topicName;
            this.numPartitions = numPartitions;
            this.replicationFactor = replicationFactor;
        }

        public String name() {
            return topicName;
        }

        public int numPartitions() {
            return numPartitions;
        }

        public int replicationFactor() {
            return replicationFactor;
        }

        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
            if (o == null || getClass() != o.getClass()) return false;
            AuditedTopic that = (AuditedTopic) o;
            return numPartitions == that.numPartitions &&
                replicationFactor == that.replicationFactor &&
                topicName.equals(that.topicName);
        }

        @Override
        public int hashCode() {
            return Objects.hash(topicName, numPartitions, replicationFactor);
        }
    }

    public enum EventType {
        CREATE, DELETE
    }

    private final Map<AuditedTopic, AuditInfo> auditInfo;
    private final EventType eventType;

    public static TopicEvent topicCreateEvent(Map<AuditedTopic, AuditInfo> auditInfo) {
        return new TopicEvent(auditInfo, EventType.CREATE);
    }

    public static TopicEvent topicDeleteEvent(Map<AuditedTopic, AuditInfo> auditInfo) {
        return new TopicEvent(auditInfo, EventType.DELETE);
    }

    public TopicEvent(Map<AuditedTopic, AuditInfo> auditInfo, EventType eventType) {
        this.auditInfo = Collections.unmodifiableMap(auditInfo);
        this.eventType = eventType;
    }

    public Map<AuditedTopic, AuditInfo> auditInfo() {
        return auditInfo;
    }

    public EventType eventType() {
        return eventType;
    }
}
```

AclEvent

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements. See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License. You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.apache.kafka.server.auditor.events;

import org.apache.kafka.common.acl.AclBinding;
import org.apache.kafka.common.acl.AclBindingFilter;
import org.apache.kafka.common.annotation.InterfaceStability;
import org.apache.kafka.server.auditor.AuditEvent;
import org.apache.kafka.server.auditor.AuditInfo;
import org.apache.kafka.server.authorizer.AclDeleteResult;

import java.util.Collections;
import java.util.Map;
import java.util.Set;

@InterfaceStability.Evolving
public class AclEvent<T, R> extends AuditEvent {

    public enum EventType {
        CREATE, DELETE, DESCRIBE;
    }

    private final AuditInfo clusterAuditInfo;
    private final Set<T> auditedEntities;
    private final EventType eventType;
    private final Map<R, Integer> operationResults;

    public static AclEvent<AclBinding, AclBinding> aclCreateEvent(Set<AclBinding> auditedEntities,
                                                                  AuditInfo clusterAuditInfo) {
        return new AclEvent<>(auditedEntities, clusterAuditInfo, Collections.emptyMap(), EventType.CREATE);
    }

    public static AclEvent<AclBinding, AclBinding> aclCreateEvent(Set<AclBinding> auditedEntities,
                                                                  AuditInfo clusterAuditInfo,
                                                                  Map<AclBinding, Integer> results) {
        return new AclEvent<>(auditedEntities, clusterAuditInfo, results, EventType.CREATE);
    }

    public static AclEvent<AclBindingFilter, AclDeleteResult> aclDeleteEvent(Set<AclBindingFilter>
auditedEntities,
                                                                             AuditInfo clusterAuditInfo) {
        return new AclEvent<>(auditedEntities, clusterAuditInfo, Collections.emptyMap(), EventType.DELETE);
    }

    public static AclEvent<AclBindingFilter, AclDeleteResult> aclDeleteEvent(Set<AclBindingFilter>
auditedEntities,
                                                                             AuditInfo clusterAuditInfo,
                                                                             Map<AclDeleteResult, Integer>
results) {
        return new AclEvent<>(auditedEntities, clusterAuditInfo, results, EventType.DELETE);
    }

    public static AclEvent<AclBindingFilter, AclBinding> aclDescribeEvent(Set<AclBindingFilter> auditedEntities,
```

```
                                                                    AuditInfo clusterAuditInfo,
                                                                    Map<AclBinding, Integer> results) {
        return new AclEvent<>(auditedEntities, clusterAuditInfo, results, EventType.DESCRIBE);
    }

    public AclEvent(Set<T> auditedEntities, AuditInfo clusterAuditInfo, Map<R, Integer> operationResults,
EventType eventType) {
        this.auditedEntities = Collections.unmodifiableSet(auditedEntities);
        this.clusterAuditInfo = clusterAuditInfo;
        this.eventType = eventType;
        this.operationResults = Collections.unmodifiableMap(operationResults);
    }

    public Set<T> auditedEntities() {
        return auditedEntities;
    }

    public AuditInfo clusterAuditInfo() {
        return clusterAuditInfo;
    }

    public Map<R, Integer> operationResults() {
        return operationResults;
    }

    public EventType eventType() {
        return eventType;
    }
}
```

Generic Events

Not all request types will need to be accompanied with the corresponding AuditEvent class as there are 50+ Kafka APIs where many are control requests which may or may not be relevant from the user's perspective and it would be very labour intensive to code and maintain these. To overcome this the auditor may pass `null` as the `AuditEvent` parameter in the audit method.

Proposed Changes

As part of the KIP we will define the interface above, implement the hooks in the various handle calls in `KafkaApis` similarly to the `Authorizer`, but doing the auditing before sending the response back as this is a common point where all the required parameters are ready. Any specific implementation will live in the respective projects as we do it with the `Authorizer`. This shouldn't be an extra burden on these specific implementations as they usually already implement the `Authorizer` or already have some client side Kafka dependencies.

Default Implementation

We will have a default logger implementation that logs the following audited events under a logger named `auditLogger`:

- Topic events: describe, list, create, delete, partition number change, replication factor change
- Config events: describe, alter config (incremental as well as legacy)
- ACL events: describe, create, delete
- replica log dirs: describe, alter
- Reassignment: alter, list
- Groups: describe, delete
- Scram credentials: describe, alter
- Client quotas: describe, alter
- Delete records
- Delete offsets

Compatibility, Deprecation, and Migration Plan

This is entirely new functionality so there are no compatibility, deprecation, or migration concerns.

Testing

The correctness of the `LoggingAuditor` and data propagation between `KafkaApis` and the `Auditor` will be covered on the unit test level with mocking.

Rejected Alternatives

Original KIP-567

There was an earlier attempt to tackle this problem but it is now abandoned. It operated with somewhat different interfaces but overall the concept was similar. I chose to take a slightly different angle and emphasize the similarities with the `Authorizer` as it makes sense to represent a similar requirement with a similar interface, therefore until the community discussion prefers otherwise, I keep the original works but represent it in the rejected alternatives.

Client Side Auditing

Some auditing action can be quite heavy, such as auditing client actions, like detecting which client produces to which topics. It was considered to do some of these on the client side but it has multiple obstacles:

- Auditing information still need to be collected in a central place, so it would require extra configuration on the client side.
- Also repetition of the same events should be avoided which means we have to implement cache on the client side. This makes the clients more heavy which we would like to avoid. Also the same caching would apply to the brokers as well so implementation-wise we wouldn't be ahead.

AbstractRequest and AbstractResponse in audit()

To provide a very generic auditing-like interface we can just pass the `AbstractRequest` and `AbstractResponse` objects to the `audit()` method call. The first problem with this is they are not public interfaces so first of all we would need to publish them as interface classes. Then the next problem that they expose a bunch of generated classes which therefore would be interfaces as well, so just by exposing these two classes we need to expose a lot of others as well, therefore growing the footprint too much. Secondly the created interface would be like a generic interceptor rather than an auditor. This is not what this KIP aims for, although the "audit" functionality could be inserted as a post-action interceptor. Refactoring the `KafkaApis` code to allow inserting interceptors would be a too big scope for this KIP.

Event Class per API

In Kafka there are 50+ APIs. To create an event for each one, like TopicCreateEvent, TopicDeleteEvent etc. would explode the boilerplate code we would need to implement the audit functionality. This isn't what we want and wouldn't be a good programming practice either. Instead of this we created event classes mostly around the resource types that are being manipulated (topics, acls, configs, etc.). These are much less in number and could be easier to use.