KIP-679: Producer will enable the strongest delivery guarantee by default

Status

Current state: *APPROVED*

Discussion thread: https://lists.apache.org/thread.html/r56f658f1d1de2b09465d70be69a8bebfd4518663be5a88fba2e9e7c0%40%3Cdev.kafka.apache.org%3E

Vote thread: https://lists.apache.org/thread.html/r1e912a4e6b6def9fbaf8e0aeb7bbcfd612f3100df31782a307268a5c%40%3Cdev.kafka.apache.org%3E

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

The producer should enable the strongest message delivery guarantee by default.

1. Exactly Once Delivery. In KIP-98, the community introduces Exactly Once Delivery, which guarantees that every message will be persisted exactly once without data loss or duplication. However, in order to give the community time for upgrading, by default, the producer will still use at-least-once delivery and set "enable.idempotence=false".
2. N-1 failure toleration. If the brokers are flushing records to disks asynchronously, the strongest concurrent failure Kafka can tolerate is N-1. (i.e. N-1 brokers shut down before the memory messages flush to disk). However, for some performance reasons, by default, the producer config will set "ack=1", where the data loss can happen if the partition leader shutdown.

Having these guarantee won't impact the performance in a significant way, as stated here:

An analysis of the impact of max.in.flight.requests.per.connection and acks on Producer performance

Public Interfaces

1. Producer config defaults will change in release version 3.0

Producer config name	From	To
enable.idempotence	false	true
acks	1	all

2. Producer ACL required for enabling idempotence will change from `IDEMPOTENT_WRITE` to `WRITE` in release version 2.8

A. In `ProduceRequest` handler, we remove the `IDEMPOTENT_WRITE` access check since it already has the `WRITE` access check on topics.

B. In `InitProduceIdRequest` handler, if the producer will get authorized if it has `WRITE` access on ANY topic.

C. `kafka-acls` will show deprecation warnings if users are trying to grant the `IDEMPOTENT_WRITE` access.

3. `org.apache.kafka.server.authorizer.Authorizer` will have a new interface for checking if the caller is authorized to perform the given ACL operation on at least one resource satisfying the filter. It will have a default implementation assuming `allow.everyone.if.no.acl.found=false`.

clients/src/main/java/org/apache/kafka/server/authorizer/Authorizer.java

```
    /**
     * Check if the caller is authorized to perform the given ACL operation on at least one
     * resource of the given type.
     *
     * 1. Filter out all the resource pattern corresponding to the requestContext, AclOperation,
     *    and ResourceType
     * 2. If wildcard deny exists, return deny directly
     * 3. For any literal allowed resource, if there's no dominant literal denied resource, and
     *    no dominant prefixed denied resource, return allow
     * 4. For any prefixed allowed resource, if there's no dominant denied resource, return allow
     * 5. For any other cases, return deny
     *
     * It is important to override this interface default in implementations because
     * 1. The interface default iterates all AclBindings multiple times, without any indexing,
     *    which is a CPU intense work.
     * 2. The interface default rebuild several sets of strings, which is a memory intense work.
     *
     * @param requestContext Request context including request resourceType, security protocol, and listener
name
     * @param op             The ACL operation to check
     * @param resourceType   The resource type to check
     * @return               Return {@link AuthorizationResult#ALLOWED} if the caller is authorized to perform
the
     *                       given ACL operation on at least one resource of the given type.
     *                       Return {@link AuthorizationResult#DENIED} otherwise.
     */
    default AuthorizationResult authorizeByResourceType(AuthorizableRequestContext requestContext, AclOperation
op, ResourceType resourceType) {
        SecurityUtils.authorizeByResourceTypeCheckArgs(op, resourceType);

        ResourcePatternFilter resourceTypeFilter = new ResourcePatternFilter(
            resourceType, null, PatternType.ANY);
        AclBindingFilter aclFilter = new AclBindingFilter(
            resourceTypeFilter, AccessControlEntryFilter.ANY);

        EnumMap<PatternType, Set<String>> denyPatterns =
                new EnumMap<PatternType, Set<String>>(PatternType.class){{
            put(PatternType.LITERAL, new HashSet<>());
            put(PatternType.PREFIXED, new HashSet<>());
        }};
        EnumMap<PatternType, Set<String>> allowPatterns =
                new EnumMap<PatternType, Set<String>>(PatternType.class){{
            put(PatternType.LITERAL, new HashSet<>());
            put(PatternType.PREFIXED, new HashSet<>());
        }};

        boolean hasWildCardAllow = false;

        KafkaPrincipal principal = new KafkaPrincipal(
            requestContext.principal().getPrincipalType(),
            requestContext.principal().getName());
        String hostAddr = requestContext.clientAddress().getHostAddress();

        for (AclBinding binding : acls(aclFilter)) {
            if (!binding.entry().host().equals(hostAddr) && !binding.entry().host().equals("*"))
                continue;

            if (!SecurityUtils.parseKafkaPrincipal(binding.entry().principal()).equals(principal)
                    && !binding.entry().principal().equals("User:*"))
                continue;

            if (binding.entry().operation() != op
                    && binding.entry().operation() != AclOperation.ALL)
                continue;
            if (binding.entry().permissionType() == AclPermissionType.DENY) {
                switch (binding.pattern().patternType()) {
                    case LITERAL:
                        if (binding.pattern().name().equals(ResourcePattern.WILDCARD_RESOURCE))
                            return AuthorizationResult.DENIED;
                        denyPatterns.get(PatternType.LITERAL).add(binding.pattern().name());
                        break;
                    case PREFIXED:
                        denyPatterns.get(PatternType.PREFIXED).add(binding.pattern().name());
                        break;
                    default:
                }
                continue;
            }

            if (binding.entry().permissionType() != AclPermissionType.ALLOW)
                continue;

            switch (binding.pattern().patternType()) {
                case LITERAL:
                    if (binding.pattern().name().equals(ResourcePattern.WILDCARD_RESOURCE)) {
                        hasWildCardAllow = true;
                        continue;
                    }
                    allowPatterns.get(PatternType.LITERAL).add(binding.pattern().name());
                    break;
                case PREFIXED:
                    allowPatterns.get(PatternType.PREFIXED).add(binding.pattern().name());
                    break;
                default:
            }
        }

        if (hasWildCardAllow) {
            return AuthorizationResult.ALLOWED;
        }

        for (Map.Entry<PatternType, Set<String>> entry : allowPatterns.entrySet()) {
            for (String allowStr : entry.getValue()) {
                if (entry.getKey() == PatternType.LITERAL
                        && denyPatterns.get(PatternType.LITERAL).contains(allowStr))
                    continue;
                StringBuilder sb = new StringBuilder();
                boolean hasDominatedDeny = false;
                for (char ch : allowStr.toCharArray()) {
                    sb.append(ch);
                    if (denyPatterns.get(PatternType.PREFIXED).contains(sb.toString())) {
                        hasDominatedDeny = true;
                        break;
                    }
                }
                if (!hasDominatedDeny)
                    return AuthorizationResult.ALLOWED;
            }
        }

        return AuthorizationResult.DENIED;
    }
```

Proposed Changes

AclAuthorizer and SimpleAclAuthorizer

AclAuthorizer and AuthorizerWrapper will override the new interface `org.apache.kafka.server.authorizer.Authorizer#authorizeAny` to

1. improve the performance
2. implement the `allow.everyone.if.no.acl.found` logic

`IDEMPOTENT_WRITE` Deprecation

Besides the public interface changes above, we will deprecate `IDEMPOTENT_WRITE` in release version 3.0 because it's kind of trivial by practice.

We are relaxing the ACL restriction from `IDEMPOTENT_WRITE` to `WRITE` earlier (release version 2.8) and changing the producer defaults later (release version 3.0) in order to give the community users enough time to upgrade their broker first. So their later client-side upgrading, which enables idempotence by default, won't get blocked by the `IDEMPOTENT_WRITE` ACL required by the old version brokers.

`IDEMPOTENT_WRITE` will be deprecated in 3.0 but won't be removed in a short term, in order to give the community enough time to upgrade their `authorizer` implementation.

`request-required-acks` option in kafka-console-producer.sh will change default to -1

In kafka-console-producer.sh, we have a option: `request-required-acks` that can configure the `acks` setting in Producer. It was originally default to 1. But after this KIP, we set the default `enable.idempotence` to true, so we have to also set the default `acks` config here to -1 for this change.

Compatibility, Deprecation, and Migration Plan

For changing the default of `acks`, there won't be any compatibility issues. But several compatibility issues may occur for changing the default of `enable.idempotence`:

1. In the scenario below, people will need to either **a)** grant the producers the `IDEMPOTENT_WRITE` access or **b)** change their producer config to explicitly disable the idempotence (set `enable.idempotence = false`).
   a. use producers with release version >= 3.0
   b. use brokers with release version < 2.8
2. In the scenario below, people will need to either **a)** upgrade their topic format version >= V2 or **b)** change their producer config to explicitly disable the idempotence (set `enable.idempotence = false`).
   a. use producers with release version >= 3.0
   b. have any topic using the message format < V2 while producers with release version >= 3.0 will produce to this topic
3. In the scenario below, people will need to implement the new `Authorizer#authorizeAny` interface
   a. use their own authorizer implementation other than `AclAuthorizer` and `SimpleAclAuthorizer`
   b. the customized authorizer support the configuration `allow.everyone.if.no.acl.found`

Rejected Alternatives

There's an alternative way to implement a fallback semantics on `enable.idempotence` to mitigate the compatibility issue. Specifically, by default, the producer will set `enable.idempotence` as `suggested`, a new option, to let the producer and brokers decide if idempotence can be enabled or not. However, since the fallback

1. adds more complexity to the client and broker side logic
2. may cause confusions and thus unexpected behavior

We decide to propose the simplest approach at the cost of the small compatibility issues.