

# KIP-590: Redirect Zookeeper Mutation Protocols to The Controller

- [Parent KIP](#)
- [Status](#)
- [Motivation](#)
- [Proposed Changes](#)
  - Existing RPCs which are sending
    - [Change How ZK Mutation Request Routs](#)
    - [Internal CreateTopicsRequest Routing](#)
- [Public Interfaces](#)
  - [Deprecate Client Side Controller Access](#)
  - [Protocol Bump](#)
  - [New Envelope RPC](#)
    - When receiving an `EnvelopeRequest`, the broker shall authorize the request with forwarding broker's principal. If the outer request is verified, the broker will continue to unwrap the inner request and handle it as normal, which means it would continue performing authorization for the inner layer principal. For KIP-590 scope, the possible top error codes are:
  - [Routing Request Security](#)
    - [Principal Serialization](#)
    - [ApiVersion Consistency](#)
    - [Routing in KIP-500](#)
  - [Monitoring Metrics](#)
- [Compatibility, Deprecation, and Migration Plan](#)
- [Rejected Alternatives](#)
- [Future Works](#)
  - [New Secure Endpoint](#)

## Parent KIP

[KIP-500: Replace ZooKeeper with a Self-Managed Metadata Quorum](#) (Accepted)


## Status

**Current state:** *Accepted*

**Discussion thread:** [here](#)

JIRAs:

 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

As part of the [KIP-500](#) initiative, we need to build a bridge release version of Kafka that could isolate the direct Zookeeper write access only to the controller. Protocols that alter cluster/topic configurations, security configurations or quotas, topics etc, should be migrated for sure as they are still relying on arbitrary broker to Zookeeper write access.

Take config change protocol for example. The current *AlterConfig* request propagation path is:

1. The admin client issues an (Incremental)AlterConfig request to broker
2. Broker updates the zookeeper path storing the metadata
3. If #2 successful, returns to the client
4. All brokers refresh their metadata upon ZK notification

Here we use ZK as the persistent storage for all the config changes, and even some brokers are not able to get in sync with ZK due to transient failures, a successful update shall be eventually guaranteed. In this KIP we would like to maintain the same level of guarantee, and make the controller as the single writer to modify the config metadata in ZK.

## Proposed Changes Existing RPCs which are sending

### Change How ZK Mutation Request Routs

The new simple routing change makes sure only controller needs to write to ZK, while other broker shall just wait for the metadata update from ZK notification eventually. As we have the source of truth configs stored in ZK still, any re-election of controller shall be safe.

For admin RPCs who are currently sending directly to the controller, brokers should support the proxy of such requests, with a revised update path during the bridge release (take `AlterConfig` as an example):

1. The admin client issues an (Incremental)`AlterConfig` request to a random broker
2. The broker redirects the request to the controller
3. The controller updates the config, and store it in ZK
4. If #3 successful, returns to the proxy broker
5. The proxy broker returns to the client as success
6. ZK update will be propagated towards all affected brokers in the cluster

This whole update strategy change would be applied to all the direct ZK mutation paths, including:

- `AlterConfig`
- `IncrementalAlterConfig`
- `CreateAcls`
- `DeleteAcls`
- `AlterClientQuotas`
- `CreateDelegationToken`
- `RenewDelegationToken`
- `ExpireDelegationToken`

Furthermore, starting from the first release version of KIP-590, the following RPCs shall be forwarded to the controller from any broker as well:

- `AlterPartitionReassignment`
- `CreatePartition`
- `CreateTopics`
- `DeleteTopics`
- `UpdateFeatures` (ongoing with KIP-584)
- `Scram`

It is because we are trying to isolate controller access from the admin client in the post KIP-500 world. Old admin clients who are sending requests directly to the controller will be given a random broker id and rely on forwarding of the original requests as well.

### Internal CreateTopicsRequest Routing

Certain edge cases we would also like to fix is for the internal topic creation.

1. `FindCoordinator` protocol has an internal topic creation logic when the cluster receives the request for the first time as transaction log topic and consumer offset topic are lazily initialized.
2. `MetadataRequest` protocol also contains an internal topic creation logic when we are looking for metadata for a non-existing internal topic and **auto-topic-creation** is enabled.

Currently the receiving broker shall just utilize its own ZK client to create internal topics, which is disallowed in the bridge release. In the post KIP world, if the broker receiving the topic creation request is the active controller, it will just handle it; otherwise, the receiving broker shall resend a new `CreateTopicRequest` to the active controller instead and let controller take care of the rest, while waiting for the response in the mean time.

One thing to note that at the moment the direct ZK access bypasses the `CreateTopicPolicy`. This is in fact a hole in the topic creation logic that we should fix. From now on, if a `MetadataRequest` tries to create an internal topic but failed, receiving broker will reply a fatal error to let the client fail fast and populate the message to the users.

## Public Interfaces

### Deprecate Client Side Controller Access

We shall remove "`ControllerNodeProvider`" on the admin client, so that new clients no longer have direct access towards the controller. All admin requests would try to use the same "`LeastLoadedNodeProvider`" to get a random node to talk to. Thus the active controller is properly isolated from the outside world, according to the [KIP-631](#).

## Protocol Bump

We also need to bump the Metadata RPC to v10 to propagate internal topic creation policy violation. Specifically:

1. For newer clients, return `POLICY_VIOLATION` when the topic creation policy is violated. In the application level, we should swap the error message with the actual failure reason such as "violation of topic creation policy when attempting to auto create internal topic through MetadataRequest."
2. For older client, return `AUTHORIZATION_FAILED` to fail the client quickly as well. It's not a perfect solution as we don't have a notification path for older clients, but at least the system admin could check for broker log when hitting this issue.

To be more strict of protecting controller information, the "ControllerId" field in new MetadataResponse shall be set to a random broker for v0-v9 request, and gets deprecated on v10. Note that only existing clients are using Metadata RPC to get controller info, so it should be safe to deprecate and we would explicitly mention that on the NetworkClient meta comments.

## New Envelope RPC

We are also going to add a new RPC type to wrap the original request during the forwarding. We will make corresponding changes to `ApiMessageTypeGenerator` class to recognize the new field `Header` and `ApiMessage` during the auto generation. And for authentication and audit logging purpose, we proposed to add the following fields:

1. Serialized Request Data
2. Id token for authentication purpose
  - a. Request principal for authentication and authorization purpose
3. Client hostname for authentication purpose

### EnvelopeRequest.json

```
{
  "apiKey": N,
  "type": "request",
  "name": "EnvelopeRequest",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "RequestData", "type": "bytes", "versions": "0+", "zeroCopy": true,
      "about": "The embedded request header and data." },
    { "name": "PrincipalIdToken", "type": "bytes", "tag": 0, "taggedVersions": "0+" },
    { "name": "RequestPrincipal", "type": "bytes", "versions": "0+", "zeroCopy": true,
      "ignorable": true, "nullableVersions": "0+", "default": "null",
      "about": "Value of the initial client principal when the request is redirected by a broker." },
    { "name": "ClientHostName", "type": "string", "versions": "0+", "default": "",
      "about": "The original client's hostname." }
  ]
}
```

When receiving an `EnvelopeRequest`, the broker shall authorize the request with forwarding broker's principal. If the outer request is verified, the broker will continue to unwrap the inner request and handle it as normal, which means it would continue performing authorization for the inner layer principal. For KIP-590 scope, the possible top error codes are:

- `NOT_CONTROLLER` as we are only forwarding admin write requests.
- `CLUSTER_AUTHORIZATION_FAILED` if the inter-broker verification failed.

The `CLUSTER` authorization for `EnvelopeRequest` takes place during the request handling, similar to `LeaderAndIsrRequest`. This ensures the `EnvelopeRequest` is not sent from a malicious client pretending to be a fellow broker. For inner request error, it will still be embedded inside the `ResponseData` struct defined in `EnvelopeResponse` below.

## EnvelopeResponse.json

```
{
  // Possible top level error code:
  //
  // NOT_CONTROLLER
  // CLUSTER_AUTHORIZATION_FAILED
  //
  "apiKey": N,
  "type": "response",
  "name": "EnvelopeResponse",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "ThrottleTimeMs", "type": "int32", "versions": "0+",
      "about": "The duration in milliseconds for which the request was throttled due to a quota violation, or
zero if the request did not violate any quota." },
    { "name": "ResponseData", "type": "bytes", "versions": "0+", "nullableVersions": "0+",
      "zeroCopy": true, "default": "null",
      "about": "The embedded response header and data." },
    { "name": "ErrorCode", "type": "int16", "versions": "0+",
      "about": "The error code, or 0 if there was no error." }
  ]
}
```

## EnvelopeResponse Handling

When the response contains NOT\_CONTROLLER error code, the forwarding broker will keep finding the correct controller until request eventually times out. For CLUSTER\_AUTHORIZATION\_FAILED, this indicates an internal error for broker security setup which has nothing to do with the client, so we have no other way but returning an UNKNOWN\_SERVER\_ERROR to the admin client.

For whatever result the controller replies to the inner request, the forwarding broker won't check. As long as the top level has no error, the forwarding broker will claim the request to be successful and reply the inner response to the admin client for the rest of error handling.

## Routing Request Security

For ZK mutation requests that need redirection, forwarding broker will just use its own authorizer to verify the principals. When the request looks good, it will just forward the request as Envelope with its own credentials, so that the controller broker will only validate the broker principal in the forwarded request. The only exceptional case is the controller audit log which needs a principal name of the request, so we will add an optional field called "InitialPrincipalName" as stated in the Envelope template.

To better understand how security check works, take AlterConfig for an example, the intended workflow for a KIP-590 broker would be:

### Step 1. Filter out resources that are authorized

- 1.1 Use traditional principals to verify first
- 1.2 If the resource is authorized, and if this is the active controller, process it
- 1.3 Otherwise package the authorized resources and send to the active controller as Envelope

### Step 2. Check the Envelope request to see if this is a forwarding request, by checking whether it sets initial principal fields and come from privileged listener

- 2.1 Use CLUSTER\_ACTION to verify, and if the resource is not authorized, return CLUSTER\_AUTHORIZATION\_FAILURE to propagate back to the original client through forwarding broker
- 2.2 if the resource is authorized but this is not the active controller, return NOT\_CONTROLLER to the sender (forwarding broker) for retry
- 2.3 Process the resource

### Step 3. Handle the returned EnvelopeResponse

- 3.1 If the top level error code is NOT\_CONTROLLER, retry until timeout
- 3.2 If the error is CLUSTER\_AUTHORIZATION\_FAILURE, set top level or resource level error code in the original RPC response.
- 3.3 Merge with other unauthorized resource and return back to the admin client

As suggested in the above process, two new error codes shall be implemented for internal authentication failure:

### Errors.java

```
BROKER_AUTHORIZATION_FAILURE(92, "Authorization failed for the request during forwarding. This indicates an internal error on the broker cluster security setup.", BrokerAuthorizationFailureException::new);
PRINCIPAL_DESERIALIZATION_FAILURE(93, "Request principal deserialization failed during forwarding. " +
"This indicates an internal error on the broker cluster security setup.",
PrincipalDeserializationFailureException::new)
```

Unfortunately for older admin clients they couldn't interpret this code, so an UNKNOWN\_SERVER\_ERROR will be presented, which is less ideal but still good enough to motivate users to check the broker side log for authorization failure. We intended to avoid returning AUTHORIZATION failure to the old client so that users don't waste time debugging any client side security setup.

To distinguish which request is forwarded, the controller will try to differentiate requests coming from inter broker listener and advertised listener. If the request is from inter broker listener, we treat it as a forwarding request and do the override authentication.

Although some users may configure the same listener name for both client and inter broker communication, which invalidates the differentiation process, this override approach still guarantees no extra security access breach since CLUSTER\_ACTION implies either the broker or a super user.

## Principal Serialization

In Kafka, principals are represented by the KafkaPrincipal type. Users are allowed to provide their own extensions through the use of a KafkaPrincipalBuilder. Extensions may include additional fields (such as a tenant ID), so we need a new mechanism to serialize and deserialize a principal. For this, we will use a new KafkaPrincipalSerde type:

### KafkaPrincipalSerde.java

```
interface KafkaPrincipalSerde {
    byte[] serialize(KafkaPrincipal principal);
    KafkaPrincipal deserialize(byte[] bytes);
}
```

Until 3.0, this type will be optional. If it is not implemented by the KafkaPrincipalBuilder type, then the broker will log a warning when the broker starts up, thus disabling the redirection as well. After 3.0, it will be required and the broker will not start without it.

Some users may not want to allow impersonation of some APIs even going beyond the limited set of supported APIs. For example, a user might prefer to allow ACL changes only from within a private network. For this use case, we extend the Authorizer so that it can distinguish impersonated requests. Specifically, we propose to add the principal that is forwarding the request to be included in the authorization context:

### AuhtorizableRequestContext.java

```
public interface AuthorizableRequestContext {
    Optional<KafkaPrincipal> forwardingPrincipal();
}
```

An Authorizer can reject impersonated requests by checking if the forwarding principal is present. This information is obviously useful for auditing as well.

## ApiVersion Consistency

Admin clients send ApiVersions to the broker upon the first connection establishes. The tricky thing after forwarding is enabled is that for forwardable APIs, admin client needs to know a commonly-agreed rang of ApiVersions among handling broker, active controller and itself.

Right now the inter-broker APIs are guaranteed by IBP constraints, but not for forwardable APIs. A compromised solution would be to put all forwardable APIs under IBP, which is brittle and hard to maintain consistency.

Instead, any broker connecting to the active controller should send an ApiVersion request from beginning, so it is easy to compute that information and send back to the admin clients upon ApiVersion request from admin. Any rolling of the active controller will trigger reconnection between broker and controller, which guarantees a refreshed ApiVersions between the two. This approach avoids the tight bond with IBP and broker could just close the connection between admin client to trigger retry logic and refreshing of the ApiVersions. Since this failure should be rare, two round-trips and timeout delays are well compensated by the less engineering work.

## Routing in KIP-500

In addition, to avoid exposing this forwarding power to the admin clients, the routing request shall be forwarded towards the controller broker internal endpoint which should be only visible to other brokers inside the cluster in the KIP-500 controller. Any admin configuration request with broker principal should not be going through the public endpoint and will be rejected for security purpose. For pre-KIP-500 controller, we would allow broker principal to go through only when the message comes in on the inter-broker listener, which is an indication of a forwarding request. The pre-KIP-500 cluster could not fully prevent malicious client pretending to be a forwarding request, but the attacker must have super user access to gain `CLUSTER_ACTION`.

## Monitoring Metrics

To effectively monitor the admin request forwarding status, we would the following metered metric:

```
MBean:kafka.server:type=RequestMetrics,name=NumRequestsForwardingToControllerPerSec,clientId=([-.\w]+)
```

to visualize how many RPC are inflight from each admin client. It will be added via [Yammer metrics](#).

## Compatibility, Deprecation, and Migration Plan

The upgrade path shall be guarded by the `inter.broker.protocol (IBP)` to make sure the routing behavior is consistent. After first rolling bounce to upgrade the binary version, all fellow brokers are still handling ZK mutation requests by themselves. With the second IBP bump rolling bounce, all upgraded brokers will be using the new routing algorithm effectively described in this KIP.

For older admin clients which still "try to" send the request to the controller, receiving broker will redirect the request to the active controller as stated in the KIP. The handling should be exactly the same no matter the ZK mutation request is from an old or a new admin client.

## Rejected Alternatives

- We discussed about the possibility of immediately building a metadata topic to propagate the changes. This seems aligned with the eventual metadata quorum path, but at a cost of blocking the current API migration towards the bridge release, since the metadata quorum design is much more complicated and requires more iterations. To avoid this extra dependency on other tracks, we should go ahead and migrate existing protocols to meet the bridge release goal sooner.
- We thought about adding an alerting metrics called **request-forwarding-to-controller-authorization-fail-count** in an effort to help administrator detect wrong security setup sooner. However, there should already be metrics monitoring request failures, so this metric could be optional.
- We thought about monitoring older client connections in the long term after bridge release, when we perform some incompatible changes to the Raft Quorum, to better capture the timing for a major version bump. However, [KIP-511](#) also has already exposed metrics like an "unknown" software name and an "unknown" software version which could serve for this purpose.
- We discussed about maintaining the client access to the controller, which has conflicts with KIP-631, so we decide to go extra steps to give existing ZK mutation RPC with forwarding power as well.
- We could embed the original request version as a tag to the forward request and always forward with the minimum version that supports flexible fields, when the original request version is older than min version to support initial principals. While dealing with the request on the controller, it will use the original version for deserialization when defined. This proposal was rejected because of the error-prone approach around the upgrade and downgrade of the RPC during transition. Generally the pattern requires all parties to parse the request and package response correctly.

## Future Works

We have also discussed about migrating the metadata read path to controller-only for read-after-write consistency. This sounds like a nice improvement but needs more discussions on trade-offs between overloading controller and the metadata consistency, also the progress of Raft quorum design as well.

## New Secure Endpoint

To maintain the same level of security going along in the post-ZK world, the broker-controller communication should have extra security guarantee. To make that happen, we will introduce a separate `ControllerEndpoint` for user to configure the exclusive access of forwarding requests to only go through this tunnel. Getting a separate communication channel also helps differentiating whether the request is from admin client or forwarded, which means the forwarding brokers don't have to bump the request version unnecessarily.

This part of the design is dependent on the Controller refactoring effort, and more details shall reveal for subsequent KIPs. It won't block the acceptance for this KIP either, since the forwarding behavior shall be the same.