KIP-4 - Command line and centralized administrative operations

The goals behind the command line shell are fundamentally to provide a centralized management for Kafka operations.

Status

Current state: *Accepted*

Discussion 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

Users of Kafka have created dozens of different systems to work with Kafka. Providing a wire protocol that allows the brokers to execute administrative code and public api/client has many benefits including:

- *Allows clients in any language to administrate Kafka*
  - *Wire protocol is supported by any language*
- *Provides public client for performing admin operations*
  - *Ensures integration test code in other projects and clients maintains compatibility*
  - *Prevents users from needing to use the Command classes and work around standard output and system exits*
- *Removing the need for admin scripts (kafka-topics.sh, kafka-acls.sh, etc) to talk directly to Zookeeper.*
  - *Allows ZNodes to be completely locked down via ACLs*
  - *Further hides the Zookeeper details of Kafka*

Public Interfaces

- Changes to **Wire Protocol**:

- Adds the following new Request/Response messages:
    - CreateTopics
    - AlterTopics
    - DeleteTopics
    - ListAcls
    - AlterAcls
    - DescribeConfig (moved to KIP-133: Describe and Alter Configs Admin APIs)
    - AlterConfig (moved to KIP-133: Describe and Alter Configs Admin APIs)
- Modifies Metadata Request/Response to allowing polling for in-progress or complete admin operations. Added fields include:
    - Add the ability to request no topics with a null topics list
    - Boolean indicating if a topic is marked for deletion
    - Boolean indicating if a topic is an internal topic
    - Rack information (if not added by KIP-36 Rack aware replica assignment)
    - Boolean indicating if a broker is the controller

Proposed Changes

Proposed changes include 2 parts:

1. Wire protocol additions and changes
2. Server-side message handlers and authorization

Follow Up Changes

Changes that should be considered shortly after or are enabled by this KIP included:

- **General**
    - New Java AdminClient implementation (KIP-117)
    - Refactor admin scripts and code to use new client where appropriate
    - Support forwarding requests to the required broker (coordinator, group leader, partition leader) (*KAFKA-1912*)
        - *See Request Forwarding below*
- **Metadata Schema**
    - Consider supporting regex topic filters in the request
    - Filter internal topics using the returned metadata (Unable to render Jira issues macro, execution error.)
- **Topic Admin Schema**
    - Improve the broker side delete topic implementation
        - Delete is likely to get used more once creation/deletion is made easier with the client. The broker side implementation has had many jiras.
        - Currently can't delete unhealthy topics.
    - Support cluster consistent blocking to wait until all relevant brokers have the required metadata
        - This may require significant re-work of the controller to do correctly
        - See Cluster Consistent Blocking below
    - Implement auto-topic creation client side (KAFKA-2410)
    - Add topic creation to the MirrorMaker client?
    - Support renaming topics (KAFKA-2333)
        - This might required unique ids for topics instead of using the name (this would improve delete too)
    - Improve reliability and speed of topic deletion
        - Support force deleting unhealthy topics
        - Support marking for deletion and async data cleanup
            - This would required unique ids for topics instead of using the name (this is needed for rename too)
            - The topic can then be marked as deleted instead of requiring all data to be removed immediately and in the mean time a new topic with the same name can be created.
- **ACL Admin Schema**
    - Review privileges for listing and altering ACLs to be more fine grained.
    - Provide an Authorizer interface using the new Java classes used by the ACL requests/responses (KAFKA-3509)
        - Deprecate the old one to encourage transition
    - Define standard Exceptions that can be thrown by the Authorizer in the interface (KAFKA-3266)
        - Otherwise all exceptions are unknown server exception to the client
    - Consider building a sync call into the Authorizer to ensure changes are propagated

Details

1. Wire Protocol Extensions

Schema

Overall the idea is to extend Wire Protocol to cover all existing admin commands so that a user does not need to talk directly to Zookeeper and all commands can be authenticated via Kafka. At the same time, since the Wire Protocol is a public API to the Kafka cluster, it was agreed that the new Admin schema needs to be "orthogonal", i.e. new messages shouldn't duplicate each other or existing requests, if those already cover particular use cases. Finally, admin requests are likely to be used not only in CLI tools, where the common use case is create/change/delete a single entity. Since Kafka is able to maintain a huge number of topics it is vital user can efficiently request many commands at one time. That's why all admin messages essentially are batch requests, i.e. it is possible to group commands of one type for many topics in one batch reducing network calls. At the same time to make Schema usage transparent and compliant with existing requests (such as Produce and Fetch) if batch request includes more than one instruction for a specific topic only the last from the list will be executed, others will be silently ignored.

New Protocol Errors

It is proposed to use existing / add these error codes to the protocol.

Error	Description
`TopicExistsException`	Topic with this name already exists
`InvalidTopic (existing)`	Topic name contains invalid characters or doesn't exist
`InvalidPartitionsException`	`Partitions` field is invalid (e.g. negative or increasing number of partitions in existing topic)
`InvalidReplicationFactorException`	`ReplicationFactor` field is invalid (e.g. negative)
`InvalidReplicaAssignmentException`	`ReplicaAssignment` field is invalid (e.g. contains duplicates)
`InvalidConfigurationException`	Configuration setting or value is incorrect
`NotControllerException`	The request was routed to a broker that wasn't the active controller
InvalidRequestException	Thrown when a request breaks basic wire protocol rules. (Existing but not mapped)

Generally, a client should have enough context to provide descriptive error message.

The same notation as in A Guide To The Kafka Protocol is used here.

Metadata Schema (Voted and Adopted in 0.10.0.0)

Metadata Request (version 1)

```
MetadataRequest => [topics]
```

Stays the same as version 0 however behavior changes.
In version 0 there was no way to request no topics, and and empty list signified all topics.
In version 1 a null topics list (size -1 on the wire) will indicate that a user wants **ALL** topic metadata. Compared to an empty list (size 0) which indicates metadata for **NO** topics should be returned.

Metadata Response (version 1)

```
MetadataResponse => [brokers] controllerId [topic_metadata]
  brokers => node_id host port rack
    node_id => INT32
    host => STRING
    port => INT32
    rack => NULLABLE_STRING
  controllerId => INT32
  topic_metadata => topic_error_code topic is_internal [partition_metadata]
    topic_error_code => INT16
    topic => STRING
    is_internal => BOOLEAN
    partition_metadata => partition_error_code partition_id leader [replicas] [isr]
      partition_error_code => INT16
      partition_id => INT32
      leader => INT32
      replicas => INT32
      isr => INT32
```

Adds rack, controller_id, and is_internal to the version 0 response.

The behavior of the replicas and isr arrays will be changed in order to support the admin tools, and better represent the state of the cluster:

- In version 0, if a broker is down the replicas and isr array will omit the brokers entry and add a REPLICA_NOT_AVAILABLE error code.
- In version 1, no error code will be set and a the broker id will be included in the replicas and isr array.
    - Note: A user can still detect if the replica is not available, by checking if the broker is in the returned broker list.

Topic Admin Schema

Create Topics Request (KAFKA-2945): (Voted and Committed for 0.10.1.0)

```
CreateTopics Request (Version: 0) => [create_topic_requests] timeout
  create_topic_requests => topic num_partitions replication_factor
[replica_assignment] [configs]
    topic => STRING
    num_partitions => INT32
    replication_factor => INT16
    replica_assignment => partition_id [replicas]
      partition_id => INT32
      replicas => INT32
    configs => config_key config_value
      config_key => STRING
      config_value => STRING
  timeout => INT32
```

`CreateTopicsRequest` is a batch request to initiate topic creation with either predefined or automatic replica assignment and optionally topic configuration.

Request semantics:

1. Must be sent to the controller broker
2. If there are multiple instructions for the same topic in one request an InvalidRequestException will be logged on the broker and a single error code for that topic will be returned to the client
    - This is because the list of topics is modeled server side as a map with TopicName as the key
3. The principal must be authorized to the "Create" Operation on the "Cluster" resource to create topics.
    - Unauthorized requests will receive a ClusterAuthorizationException
4. Only one from `ReplicaAssignment or (num_partitions +` replication_factor`),` can be defined in one instruction.

    - If both parameters are specified an InvalidRequestException will be logged on the broker and an error code for that topic will be returned to the client
    - In the case `ReplicaAssignment` is defined number of partitions and replicas will be calculated from the supplied replica_assignment.
    - In the case of defined (num_partitions + replication_factor) replica assignment will be automatically generated by the server.
    - One or the other must be defined. The existing broker side auto create defaults will not be used (default.replication.factor, num. partitions). The client implementation can have defaults for these options when generating the messages.
    - The first replica in [replicas] is assumed to be the preferred leader. This matches current behavior elsewhere.
5. Setting a timeout > 0 will allow the request to block until the topic metadata is "complete" on the controller node.
    - Complete means the local topic metadata cache been completely populated and all partitions have leaders
        - The topic metadata is updated when the controller sends out update metadata requests to the brokers
    - If a timeout error occurs, the topic could still be created successfully at a later time. Its up to the client to query for the state at that point.
6. Setting a timeout <= 0 will validate arguments and trigger the create topics and return immediately.
    - This is essentially the fully asynchronous mode we have in the Zookeeper tools today.
    - The error code in the response will either contain an argument validation exception or a timeout exception. If you receive a timeout exception, because you asked for 0 timeout, you can assume the message was valid and the topic creation was triggered.
7. The request is not transactional.
    a. If an error occurs on one topic, the others could still be created.
    b. Errors are reported independently.

QA:

- Why is CreateTopicsRequest a batch request?
    - Scenarios where tools or admins want to create many topics should be able to with fewer requests
    - Example: MirrorMaker may want to create the topics downstream
- What happens if some topics error immediately? Will it return immediately?
    - The request will block until all topics have either been created, errors, or the timeout has been hit
    - There is no "short circuiting" where 1 error stops the other topics from being created
- Why implement "partial blocking" instead of fully async or fully consistent?
    - See Cluster Consistent Blocking below
- Why require the request to go to the controller?
    - The controller is responsible for the cluster metadata and its propagation
    - See Request Forwarding below

Create Topics Response

```
CreateTopics Response (Version: 0) => [topic_error_codes]
  topic_error_codes => topic error_code
    topic => STRING
    error_code => INT16
```

`CreateTopicsResponse` contains a map between topic and topic creation result error code (see New Protocol Errors).

Response semantics:

1. When a request hits the timeout, the topics that are not "complete" will have the TimeoutException error code.
    - The topics that did complete successfully with have no error.

Delete Topics Request (KAFKA-2946): (Voted and Planned for 0.10.1.0)

```
DeleteTopics Request (Version: 0) => [topics] timeout
  topics => STRING
  timeout => INT32
```

DeleteTopicsRequest is a batch request to initiate topic deletion.

`Request semantics:`

1. Must be sent to the controller broker
2. If there are multiple instructions for the same topic in one request the extra request will be ingnored
    - This is because the list of topics is modeled server side as a set
    - Multiple deletes results in the same end goal, so handling this error for the user should be okay
3. When requesting to delete a topic that does not exist, a an InvalidTopic error will be returned for that topic.
4. When requesting to delete a topic that is already marked for deletion, the request will wait up to the timeout until the delete is "complete" and return as usual.
    - This is to avoid errors due to concurrent delete requests. The end result is the same, the topic is deleted.
5. The principal must be authorized to the "Delete" Operation on the "Topic" resource to delete the topic.
    - Unauthorized requests will receive a TopicAuthorizationException if they are authorized to the "Describe" Operation on the "Topic" resource
    - Otherwise they will receive an InvalidTopicException as if the topic does not exist.
6. Setting a timeout > 0 will allow the request to block until the delete is "complete" on the controller node.
    - Complete means the local topic metadata cache no longer contains the topic
        - The topic metadata is updated when the controller sends out update metadata requests to the brokers
    - If a timeout error occurs, the topic could still be deleted successfully at a later time. Its up to the client to query for the state at that point.
7. Setting a timeout <= 0 will validate arguments and trigger the delete topics and return immediately.
    - This is essentially the fully asynchronous mode we have in the Zookeeper tools today.
    - The error code in the response will either contain an argument validation exception or a timeout exception. If you receive a timeout exception, because you asked for 0 timeout, you can assume the message was valid and the topic deletion was triggered.
8. The request is not transactional.
    a. If an error occurs on one topic, the others could still be deleted.
    b. Errors are reported independently.

QA:

- Why is DeleteTopicsRequest a batch request?
    - Scenarios where tools or admins want to delete many topics should be able to with fewer requests
    - Example: Removing all cluster topics
- What happens if some topics error immediately? Will it return immediately?
    - The request will block until all topics have either been deleted, errors, or the timeout has been hit
    - There is no "short circuiting" where 1 error stops the other topics from being deleted
- Why have a timeout at all? Deletes could take a while?
    - True some deletes may take a while or never finish, however some admin tools may want extended blocking regardless.
    - If you don't want any blocking setting a timeout of 0 works.
    - Future changes may make deletes much faster. See the Follow Up Changes section above.
- Why implement "partial blocking" instead of fully async or fully consistent?
    - See Cluster Consistent Blocking below
- Why require the request to go to the controller?
    - The controller is responsible for the cluster metadata and its propagation
    - See Request Forwarding below

Delete Topics Response

```
DeleteTopics Response (Version: 0) => [topic_error_codes]
  topic_error_codes => topic error_code
    topic => STRING
    error_code => INT16
```

`DeleteTopicsResponse` contains a map between topic and topic creation result error code (see New Protocol Errors).

Response semantics:

1. When a request hits the timeout, the topics that are not "complete" will have the TimeoutException error code.
    - The topics that did complete successfully with have no error.

Alter Topics Request

This request/response is a bit more complicated and less critical than some others. Therefore, It will be addressed toward the end of KIP-4.

ACL Admin Schema (KAFKA-3266)

Note: Some of this work/code overlaps with "KIP-50 - Move Authorizer to o.a.k.common package". KIP-4 does not change the Authorizer interface at all, but does provide java objects in "org.apache.kafka.common.security.auth" to be used in the protocol request/response classes. It also provides translations between the Java and Scala versions for server side compatibility with the Authorizer interface.

List ACLs Request

```
ListAcls Request (Version: 0) => principal resource
  principal => NULLABLE_STRING
  resource => resource_type resource_name
    resource_type => INT8
    resource_name => STRING
```

Request semantics:

1. Can be sent to any broker
2. If a non-null principal is provided the returned ACLs will be filtered by that principal, otherwise ACLs for all principals will be listed.
3. If a resource with a resource_type != -1 is provided ACLs will be filtered by that resource, otherwise ACLs for all resources will be listed.
4. Any principal can list their own ACLs where the permission type is "Allow", Otherwise the principal must be authorized to the "All" Operation on the "Cluster" resource to list ACLs.
    - Unauthorized requests will receive a ClusterAuthorizationException
    - This avoids adding a new operation that an existing authorizer implementation may not be aware of.
    - This can be reviewed and further refined/restricted as a follow up ACLs review after this KIP. See Follow Up Changes.
5. Requesting a resource or principal that does not have any ACLs will not result in an error, instead empty response list is returned

List ACLs Response

```
ListAcls Response (Version: 0) => [responses] error_code
  responses => resource [acls]
    resource => resource_type resource_name
      resource_type => INT8
      resource_name => STRING
    acls => acl_principal acl_permission_type acl_host acl_operation
      acl_principal => STRING
      acl_permission_type => INT8
      acl_host => STRING
      acl_operation => INT8
  error_code => INT16
```

Alter ACLs Request

```
AlterAcls Request (Version: 0) => [requests]
  requests => resource [actions]
    resource => resource_type resource_name
      resource_type => INT8
      resource_name => STRING
    actions => action acl
      action => INT8
      acl => acl_principal acl_permission_type acl_host acl_operation
        acl_principal => STRING
        acl_permission_type => INT8
        acl_host => STRING
        acl_operation => INT8
```

Request semantics:

1. Must be sent to the controller broker
2. If there are multiple instructions for the same resource in one request an InvalidRequestException will be logged on the broker and a single error code for that resource will be returned to the client
    - This is because the list of requests is modeled server side as a map with resource as the key
3. ACLs with a delete action will be processed first and the add action second.
    a. This is to prevent confusion about sort order and final state when a batch message is sent.

    b. If an add request was processed first, it could be deleted right after.
    c. Grouping ACLs by their action allows batching requests to the authorizer via the Authorizer.addAcls and Authorizer.removeAcls calls.
4. The request is not transactional. One failure wont stop others from running.
    a. If an error occurs on one action, the others could still be run.
    b. Errors are reported independently.
5. The principal must be authorized to the "All" Operation on the "Cluster" resource to alter ACLs.
    - Unauthorized requests will receive a ClusterAuthorizationException
    - This avoids adding a new operation that an existing authorizer implementation may not be aware of.
    - This can be reviewed and further refined/restricted as a follow up ACLs review after this KIP. See Follow Up Changes.

QA:

- Why doesn't this request have a timeout and implement any blocking like the CreateTopicsRequest?
    - The Authorizer implementation is synchronous and exposes no details about propagating the ACLs to other nodes.
    - The best we can do in the existing implementation is call Authorizer.addAcls and Authorizer.removeAcls and hope the underlying implementation handles the rest.
- What happens if there is an error in the Authorizer?
    - Currently the best we can do is log the error broker side and return a generic exception because there are no "standard" exceptions defined in the Authorizer interface to provide a more clear code
    - KIP-50 is tracking adding the standard exceptions
    - The Authorizer interface also provides no feedback about individual ACLs when added or deleted in a group
        - Authorizer.addAcls is a void function, the best we can do is return an error for all ACLs and let the user check the current state by listing the ACLs
        - Autohrizer.removeAcls is a boolean function, the best we can do is return an error for all ACLs and let the user check the current state by listing the ACLs
        - Behavior here could vary drastically between implementations
        - I suggest this be addressed in KIP-50 as well, though it has some compatibility concerns.
- Why require the request to go to the controller?
    - The controller is responsible for the cluster metadata and its propagation
    - This ensures one instance of the Authorizer sees all the changes and reduces concurrency issues, especially because the Authorizer interface exposes no details about propagating the ACLs to other nodes.
    - See Request Forwarding below

Alter ACLs Response

```
AlterAcls Response (Version: 0) => [responses]
  responses => resource [results]
    resource => resource_type resource_name
      resource_type => INT8
      resource_name => STRING
    results => action acl error_code
      action => INT8
      acl => acl_principal acl_permission_type acl_host acl_operation
        acl_principal => STRING
        acl_permission_type => INT8
        acl_host => STRING
        acl_operation => INT8
      error_code => INT16
```

2. Server-side Admin Request handlers

At the highest level, admin requests will be handled on the brokers the same way that all message types are. However, because admin messages modify cluster metadata they should be handled by the controller. This allows the controller to propagate the changes to the rest of the cluster. However, because the messages need to be handled by the controller does not necessarily mean they need to be sent directly to the controller. A message forwarding mechanism can be used to forward the message from any broker to the correct broker for handling.

Because supporting all of this is quite the undertaking I will describe the "ideal functionality" and then the "intermediate functionality" that gets us some basic administrative support quickly while working towards the optimal state.

Ideal Functionality:

1. A client sends an admin request to **any** broker
2. The admin request is forwarded to the required broker (likely the controller)
3. The request is handled and the server blocks until a timeout is reached or the requested operation is completed (failure or success)
    a. An operation is considered complete/successful when **all required nodes have the correct/current state**.
    b. Immediate follow up requests to **any broker** will succeed.
    c. Requests that timeout may still be completed after the timeout. The users would need to poll to check the state.
4. The response is generated and forwarded back to the broker that received the request.
5. A response is sent back to the client.

Intermediate Functionality:

1. A client sends an admin **write** requests to **the controller** broker. **Read** requests can still go to **any** broker.
    a. As a follow up request forwarding can be added transparently. (see below)
2. The request is handled and the server blocks until a timeout is reached or the requested operation is completed (failure or success)
    a. An operation is considered complete/successful when **the controller node has the correct/current state.**

    b. Immediate follow up requests to **the controller** will succeed. Others (not to the controller) are likely to succeed or cause a retriable exception that would eventually succeed.
    c. Requests that timeout may still be completed after the timeout. The users would need to poll to check the state.
3. A response is sent back to the client.

The ideal functionality has 2 features that are more challenging initially. For that reason those features will be removed from the initial changes, but will be tracked as follow up improvements. However, this intermediate solution should allow for a relatively transparent transition to the ideal functionality.

Request Forwarding: *KAFKA-1912*

Request forwarding is relevant to any message the needs to be sent to the "correct" broker (ex: partition leader, group coordinator, etc). Though at first it may seam simple it has many technicall challenges that need to be decided in regards to connections, failure, retries, etc. Today, we depend on the client to choose the correct broker and clients that want to utilize the cluster "optimally" would likely continue to do so. For those reasons it can be handled it can be handled generically as an independent feature.

Cluster Consistent Blocking:

Blocking an admin request until the entire cluster is aware of the correct/current state is difficult based on Kafka's current approach for propagating metadata. This approach varies based on the the metadata changing.

- Topic metadata changes are propagated via UpdateMetadata and LeaderAndIsr requests
- Config changes are propagated via zookeeper and listeners
- ACL changes depend on the implementation of the Authorizer interface
    - The default SimpleACLAuthorizer uses zookeeper and listeners

Though all of these mechanisms are different, they are all commonly "eventually consistent". None of the mechanisms, as currently implemented, will block until the metadata has been propagated successfully. Changing this behavior would require a large amount of change to the KafkaController, additional inter-broker messages, and potentially a change to the Authorizer interface. These are are all changes that should not block the implementation of KIP-4.

The intermediate changes in KIP-4 should allow an easy transition to "complete blocking" when the work can be done. This is supported by providing **optio nal** local blocking in the mean time. This local blocking only blocks until the local state on the controller is correct. We will still provide a polling mechanism for users that do not want to block at all. A polling mechanism is required in the optimal implementation too because users still need a way to check state after a timeout occurs because operations like "create topic" are not transactional. Local blocking has the added benefit of avoiding wasted poll requests to other brokers when its impossible for the request to be completed. If the controllers state is not correct, then the other brokers cant be either. Clients who don't want to validate the entire cluster state is correct can block on the controller and avoid polling all together with reasonable confidence that though they may get a retriable error on follow up requests, the requested change was successful and the cluster will be accurate eventually.

Because we already add a timeout field to the requests wire protocols, changing the behavior to block until the cluster is consistent in the future would not require a protocol change. Though the version could be bumped to indicate a behavior change.

Compatibility, Deprecation, and Migration Plan

Rejected Alternatives

If there are alternative ways of accomplishing the same thing, what were they? The purpose of this section is to motivate why the design is the way it is and not some other way.

TopicMetadataRequest/Response: After some debate we decided not to evolve the TopicMetadataResponse to remove the ISR field (which currently can return incorrect information). There is a use-case for this in KAFKA-2225, so we will treat this a bug and fix it going forward. See KAFKA-1367 for more details