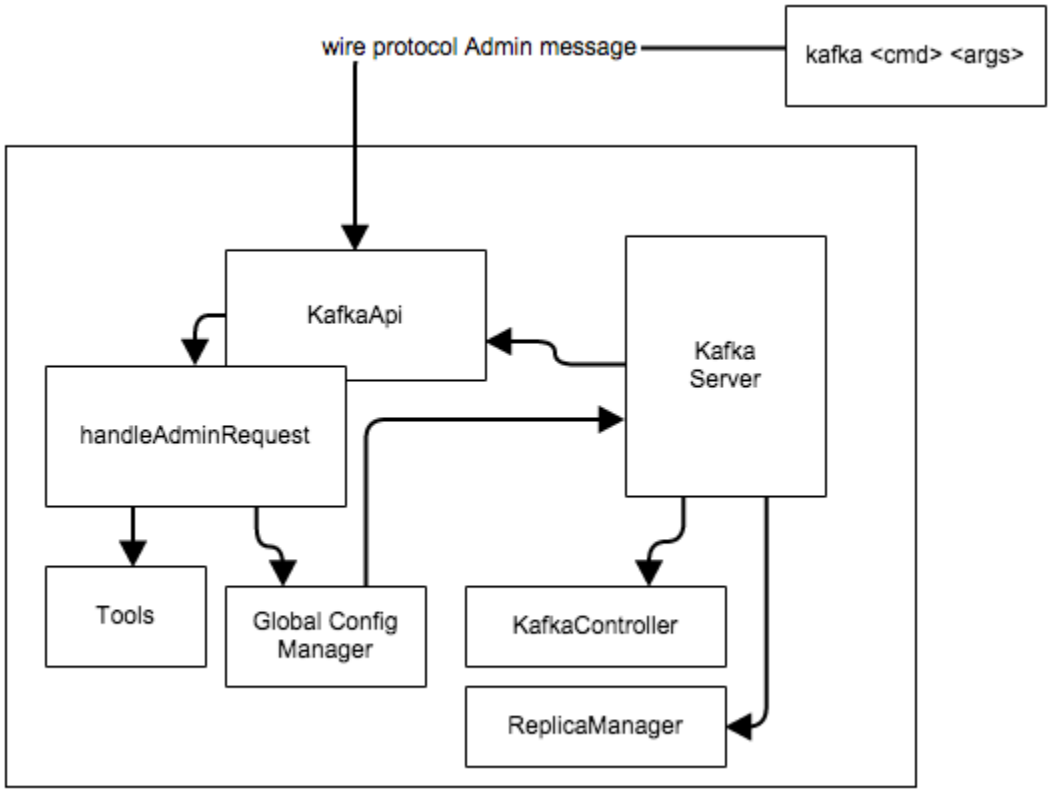
Kafka Command Line and Related Improvements

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

There are a lot of different kafka tools. Right now I think 5-6 of them are being used commonly. I was thinking we could start by taking https://cwiki.apache.org/confluence/display/KAFKA/System+Tools and https://cwiki.apache.org/confluence/display/KAFKA/Replication+tools and exposing them in a plugin type way for a command line shell interface. This would also include a new global broker configuration management and access to the tools we already have outside of the scripts both through a new wire protocol message type.



1) We need to add a new Admin message to the wire protocol that will be able to deal with passing the command line utility calls to the tools and global configuration manager on the broker (any broker). Any tool performing a task will (should be able to, need to flesh this out more) be able to execute but instead of on the command line will be on a broker thread. The controller will continue to-do the tasks it is doing today such as "create topic" however; the TopicCommand will be called from within the handleAdminRequestTools.

2) We need to implement the handleAdminRequest.

3) We need to build a client for the wire protocol. I think should be a simple CLI

It would be both: command line and shell.

so

kafka -b brokerlist -a reasign-partition status

would run from the cli and

kafka shell -b brokerlist
kafka>describe;
... kafka-topics.sh --describe
kafka>set topic_security['pci','profile','dss'] = true
...etc

An important item is that folks that are using existing tools we should have an easy api type way so they can keep doing that and get benefit too.

This interface should also have some monitoring stats too, e.g visualize in text the consumer lag trending between offset committed and log end offset.

kafka>use topic name;
kafka>stats;

not sure right this minute if we should do this in python, java or scala. I think whoever works on it can decide we can support accross the committers I think whatever it is. My thoughts are it should be in the ./clients folder.

... we "may" also want to have the CLI expose and run via HTTP REST too however; I think this can be quickly easily done by someone simply if we build it right.

4) The Global Configuration Manager. This is VERY important to the goals of these changes. MANY configuration (much like topic level) are actually global to EVERY broker and brokers (through server.properties) should be able to override but the "default" should come from storage (for now zookeeper).

a) setting this goes through the CLI and the handleAdminRequest()

b) using this is layered like this ... within KafkaApi we implement a new class that can flatten/figure out the right property. Check server.properties (we can make that level 1) and if not found use what we get from storage (for now zookeeper) and if not found then use what is default in the code. This will be very nice because you can set a default for EVERY broker for xyz configuration and not have to manage it accorss brokers with properties file. Centralized configuration 🙂

The top level for this work will be https://issues.apache.org/jira/browse/KAFKA-1694 and broken into sub tickets.

Potential Gotchas

- using RQ/RP wire protocol to the controller instead of the current way (via ZK admin path) may expose concurrency on the admin requests, which may not be supported yet. https://issues.apache.org/jira/browse/KAFKA-1305

Proposed RQ/RP Format

For each type of Admin Request a separate type of Wire protocol message is created.

Currently there are 5 types of messages which support TopicCommand - `CreateTopic(Request | Response)`, `AlterTopic`, `DeleteTopic`, `DescribeTopic`, `ListTopics`. And a special message type to identify cluster info - `ClusterMetadata` (read Kafka Admin Command Line Internals for details).

The same notation as in A Guide To The Kafka Protocol is used here. The only difference - new Kafka Protocol metatype - `MaybeOf` (`"?"` in notation), when used means value is optional in message. To define value existence special control byte is prepended before each value (`0` - field is absent, otherwise - read value normally).

Cluster Metadata Request

```
ClusterMetadataRequest
=>
```

Cluster Metadata Response

```
ClusterMEtadataResponse => ErrorCode [Broker] ?
(Controller)
  ErrorCode => int16
  Broker => NodeId Host Port
    NodeId => int32
    Host => string
    Port => int32
Controller => Broker
```

`ClusteMetadataRequest` is a request with no arguments.

`ClusterMetadataResponse` holds error code (`0` in case of successful result), list of brokers in cluster and optionally broker serving a Controller's role (returning empty Controller most likely means either error during request processing or cluster being in some intermediate state).

Admin RQ/RP format

All admin messages listed below are required to be sent only to Controller broker. Only controller will process such messages. If Admin message is sent to an ordinary broker a special error code is returned (code 22). In case of other failure during processing message `AdminRequestFailedError` is returned.

Error	Code	Description
AdminRequestFailed	21	Unexpected error occurred while processing Admin request.
InvalidRequestTarget	22	Target broker (id=<this_broker_id>) is not serving a controller's role.

Create Topic Request

```
CreateTopicRequest => TopicName ?(Partitions) ?(Replicas) ?(ReplicaAssignment)
[Config]
  TopicName => string
  Partitions => int32
  Replicas => int32
  ReplicaAssignment => string
  Config => string
```

Create Topic Response

```
CreateTopicResponse => ErrorCode ?
(ErrorDescription)
  ErrorCode => int16
  ErrorDescription => string
```

`CreateTopicRequest` requires topic name and either (partitions+replicas) or replicas assignment to create topic (validation is done on server side). You can also specify topic-level configs to create topic with (to use default set an empty array), format `key=value`.

`CreateTopicResponse` is fairly simple - you receive error code (`0` as always identifies `NO_ERROR`) and optionally error description. Usually it will hold the higher level exception that happened during command execution.

Alter Topic Request

```
AlterTopicRequest => TopicName ?(Partitions) ?(ReplicaAssignment) [AddedConfig] [DeletedConfig]
  TopicName => string
  Partitions => int32
  Replicas => int32
  AddedConfig => string
  DeletedConfig => string
```

Alter Topic Response

```
AlterTopicResponse => ErrorCode ?
(ErrorDescription)
  ErrorCode => int16
  ErrorDescription => string
```

`AlterTopicRequest` is similar to previous, to specify topic level settings that should be removed, use `DeletedConfig` array (just setting keys).

`AlterTopicResponse` is similar to `CreateTopicResponse`.

Delete Topic Request

```
DeleteTopicRequest =>
TopicName
  TopicName => string
```

Delete Topic Response

```
DeleteTopicResponse => ErrorCode ?
(ErrorDescription)
  ErrorCode => int16
  ErrorDescription => string
```

`DeleteTopicRequest` requires only topic name which should be deleted.

`DeleteTopicResponse` is similar to `CreateTopicResponse`.

Describe Topic Request

```
DescribeTopicRequest =>
TopicName
  TopicName => string
```

Describe Topic Response

```
DescribeTopicResponse => ErrorCode ?(ErrorDescription) ?(TopicDescription)
  ErrorCode => int16
  ErrorDescription => string
  TopicDescription => TopicName TopicConfigDetails [TopicPartitionDetails]
    TopicName => string
    TopicConfigDetails => Partitions ReplicationFactor [Config]
      Partitions => int32
      ReplicationFactor => int32
      Config => overridden topic-level configs
    TopicPartitionsDetails => PartitionId ?(Leader) [Replica] [ISR]
      PartitionId => int32
      Leader => int32
      Replica => int32
      ISR => int32
```

`DescribeTopicRequest` requires only topic name.

`DescribeTopicResponse` besides errorCode and optional errorDescription which are used in the same way as in previous messages, holds optional (non empty if execution was successful) `TopicDescription` structure. Its structure is the following:

Field	Description
TopicName	The name of the topic for which description is provided.
TopicConfigDetails	A structure that holds basic replication details.
Partitions	Number of partitions in give topic.
Config	Topic-level setting and value which was overridden.
TopicPartitionDetails	List describing replication details for each partition.
PartitionId	Id of the partition.
Leader	Optional broekr-leader id for the described partition.
Replicas	List of broker ids serving a replica's role for the partition.
ISR	Same as replicas but includes only brokers that are known to be "in-sync"

List Topics Request

```
ListTopicsRequest
=>
```

List Topics Response

```
ListTopicsResponse => ErrorCode ?(ErrorDescription) ?(TopicsList)
  ErrorCode => int16
  ErrorDescription => string
  TopicsList => [TopicMarkedForDeletion] [AliveTopic]
    TopicMarkedForDeletion => string
    AliveTopic => string
```

`ListTopicsRequest` is a request with no arguments.

`ListTopicsResponse` besides errorCode and optional errorDescription which are used in the same way as in previous messages, holds optional (non empty if execution was successful) two list of topic names - one for deleted topics (marked for deletion) and the second one for ordinary, alive topics.