KIP-226 - Dynamic Broker Configuration

Status

Current state: *Accepted*

Discussion thread: *here [Change the link from the KIP proposal email archive to your own email thread]*

JIRA: *KAFKA-6240*

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Kafka brokers are configured using properties (typically provided in a file named *`server.properties`*) that cannot be updated without restarting the broker. Broker configuration options can be described using the new *AdminClient*, but the configuration cannot currently be altered.

There are several use cases where dynamic updates of broker configs will help to avoid time-consuming restart of brokers. For example:

1. Update short-lived SSL keystores on the broker
2. Performance tuning based on metrics (e.g. increase network/IO threads)
3. Add, remove or reconfigure metrics reporters
4. Update configuration of all topics consistently across the whole cluster (e.g. `unclean.leader.election.enable`)
5. Update log cleaner configuration for tuning
6. Update listener/security configuration

Goals:

1. Use existing dynamic configuration mechanism to make commonly updated broker configs dynamic so that they can be updated without restarting the broker.
2. Support dynamic default configs that are picked up by all brokers in the cluster, to enable consistent configuration across all brokers (e.g. `unclean.leader.election.enable`)
3. Support individual broker config overrides for all dynamic configs to enable testing on one broker before the changes are picked up by all brokers.
4. Enable broker configs to be described and altered using *`AdminClient`*.
5. Continue to support simple property file based configuration so that brokers can be easily started up without any additional configuration steps.

In the first phase of the implementation, commonly updated broker properties will be made dynamic based on known use cases. The goal will be to make more configs dynamic later if required.

Public Interfaces

Dynamic configuration

Dynamic configs for the entity type *brokers* will be added for a limited set of configuration options by extending the existing dynamic replication quota config for brokers. The corresponding static config in *server.properties* will be continue to be supported to enable quick start of Kafka brokers with a properties file. Where applicable, default cluster-wide configs can be configured to allow consistent values to be used across all brokers (stored in ZK at */config/brokers/<default>*). For per-broker configs, broker-specific config overrides can be configured (stored in ZK at */config/brokers/id*).

The precedence used for configs will be:

1. Dynamic broker-specific config from */config/brokers/id*
2. Dynamic cluster-wide default config from */config/brokers/<default>*
3. Static config from *server.properties*

Individual broker config overrides will be provided for all the dynamic configs to enable testing. *AdminClient* will return all configured configs and defaults in the order of precedence in *DescribeConfigsResponse* if synonyms are requested. For configs that we want to keep consistent across all brokers, *kafka-configs.sh* and *AdminClient* will allow updates of cluster-wide defaults, but it will still be possible to configure per-broker overrides for testing. Brokers will handle all configs consistently with the precedence specified above.

Dynamic broker configs will be stored in ZooKeeper in JSON format along with the existing replication quota config for entity type *brokers*. With the current dynamic configuration implementation, brokers watch configuration update node in ZooKeeper and invoke config handlers when update notifications are received. The config handler for *brokers* will be updated to handle *KafkaConfig* changes as well.

The existing ACL for `AlterConfigs` operation on `Cluster` resource will be applied for broker config updates. As with topic configuration updates, *alter.config.policy.class.name* can be configured to specify a custom *AlterConfigsPolicy.*

New Broker Configuration Option

sasl.jaas.config will be supported for brokers to enable dynamic SASL configuration of brokers. The property will use the same format as clients with one login context specified as the config value. The mechanism name corresponding to the config must be specified as prefix in lower-case (e.g. `scram-sha-256.sasl.jaas.config`). If multple mechanisms are enabled on a listener, separate configs must be provided for each enabled mechanism. The property may be preceded by listener name if multiple listeners are configured to use SASL. (e.g `listener.name.sasl_ssl.plain.sasl.jaas.config`)

Format: One login context entry using the same format JAAS configuration:

```
<LoginModuleClass> <ControlFlag> *(<OptionName>=<OptionValue>);

ControlFlag = required|requisite|sufficient|optional
```

Example:

sasl.jaas.config example

```
listener.name.sasl_ssl.plain.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
user_alice="alice-secret";
listener.name.sasl_ssl.scram-sha-256.sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule
required;
```

Securing passwords in ZooKeeper

A new broker configuration will be added to specify a secret key that is used to encrypt passwords stored in ZooKeeper. The `SecretKeyFactory` and *Cipher* algorithms as well as the iteration count used will be also be made configurable. The actual algorithms, salt and iteration count will be stored along with the encrypted password to ensure that the password can be decrypted even if the encryption parameters are altered (though these configs are not being made dynamic at the moment).

- Name: *password.encoder.secret* Type: *Password*
- Name: *password.encoder.old.secret* Type: *Password (only used when password.encoder.secret is rotated)*
- Name: *password.encoder.keyfactory.algorithm* Type: *String* Default: *PBKDF2WithHmacSHA512* if available, otherwise *PBKDF2WithHmacSHA1* (e.g. Java7)
- Name: *password.encoder.cipher.algorithm* Type: *String* Default: *AES/CBC/PKCS5Padding* (AES-256 if available, AES-128 otherwise)
- Name: *password.encoder.key.length* Type: Integer Default: *256* if supported, *128* otherwise
- Name: *password.encoder.iterations* Type: *Integer* Default: *4096*

The secret will not be dynamically configurable and hence will never be stored in ZooKeeper. All the dynamic password configs are per-broker configs and hence there is no requirement to maintain the same secret across all brokers. To change *password.encoder.secret*, each broker must be restarted with an updated *server.properties* that contains the new secret in the config *password.encoder.secret* as well as the old secret in the config *password.encoder.old.secret*. The broker will decode all passwords in ZooKeeper using *password.encoder.old.secret* and update the values in ZooKeeper after re-encoding using *password.encoder.secret.* The config *password.encoder.old.secret* will be used only if the passwords in ZooKeeper are encoded using the old value and will be ignored otherwise.

Broker configuration in ZooKeeper will be protected using ACLs and will no longer be world-readable by default. It is expected that secure deployments of Kafka will also use network segmentation to limit ZooKeeper access.

Protocol Changes

A new option *include_synonyms* will be added to *DescribeConfigsRequest* to return all the configs which may be used as the value of the specified config if the config was removed. For example, *flush.ms* config for a topic will return the broker config *log.flush.interval.ms* as a synonym if *include_synonyms*=true. *DescribeConfigsResponse* will be return all synonyms in the order of precedence. This is particularly useful to obtain the default value that the config will revert to if a config override is removed. As dynamic broker configs are being added at per-broker and cluster-default levels, *include_synonyms* can be useful to list all the configured values and the precedence used to obtain the currently configured value.

DescribeConfigsRequest version will be bumped up to 1.

DescribeConfigsRequest

```
DescribeConfigs Request (Version: 1) => [resource [config_name]] include_synonyms
  resource => resource_type resource_name
    resource_type => INT8
    resource_name => STRING
  config_name => STRING
  include_synonyms => BOOLEAN

DescribeConfigs Response (Version: 1) => throttle_time_ms [entities]
  throttle_time_ms => INT32
  entities => error_code error_message resource [configs]
    error_code => INT16
    error_message => STRING
    resource => resource_type resource_name
      resource_type => INT8
      resource_name => STRING
    configs => [config_entry [synonym]]    <= Added [synonym]
    config_entry =>
      config_name => STRING
      config_value => NULLABLE_STRING
      read_only => BOOLEAN
      config_source => INT8                <= Replaced boolean is_default with more generic config_source (see
below for values)
      is_sensitive => BOOLEAN
    synonym =>                             <= NEW
      config_name => STRING
      config_value => NULLABLE_STRING
      config_source => INT8                <= may be one of
(TOPIC|DYNAMIC_BROKER|DYNAMIC_DEFAULT_BROKER|STATIC_BROKER|DEFAULT)
```

When `MetadataRequest` version is increased after 1.1.0 release, a new error code `ENDPOINT_NOT_FOUND_ON_LEADER` will be added to notify clients when a listener is available on the broker used to obtain metadata, but not on the leader of a partition. This could be a transient error when listeners are added and will be retried in the same way as `LEADER_NOT_AVAILABLE`. Broker will continue to return `LEADER_NOT_AVAILABLE` to clients using older version of `MetadataRequest`. In 1.1.0, brokers will return `LEADER_NOT_AVAILABLE` instead of `UNKNOWN_SERVER_ERROR` in older versions.

Public Interface Changes

A new interface *Reconfigurable* will be added to notify reconfigurable objects of configuration changes. For example, metrics reporters that support reconfiguration can implement the interface *Reconfigurable* to enable reconfiguration without broker restart. The interface will also be implemented by all internal classes which support reconfiguration (e.g. *ChannelBuilder*)

Reconfigurable

```
package org.apache.kafka.common;

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

/**
 * Interface for reconfigurable classes that support dynamic configuration.
 */
public interface Reconfigurable extends Configurable {

    /**
     * Returns the names of configs that may be reconfigured.
     */
    Set<String> reconfigurableConfigs();

    /**
     * Validates the provided configuration. The provided map contains
     * all configs including any reconfigurable configs that may be different
     * from the initial configuration.
     */
    boolean validateReconfiguration(Map<String, ?> configs);

   /**
     * Reconfigures this instance with the given key-value pairs. The provided
     * map contains all configs including any reconfigurable configs that
     * may have changed since the object was initially configured using
     * {@link Configurable#configure(Map)}.
     */
    void reconfigure(Map<String, ?> configs);

}
```

The classes *`DescribeConfigsOptions`* and *`DescribeConfigsResult`* used by *`AdminClient`* will be updated to include config synonyms in the result.

New methods in DescribeConfigOptions

```
/**
 * Return true if synonym configs should be returned in the response.
 */
public boolean includeSynonyms() {
    return includeSynonyms;
}

/**
 * Set to true if synonym configs should be returned in the response.
 */
public DescribeConfigsOptions includeSynonyms(boolean includeSynonyms) {
    this.includeSynonyms = includeSynonyms;
    return this;
}
```

New methods in ConfigEntry

```
/**
 * Returns all config values that may be used as the value of this config along with their source,
 * in the order of precedence. The list starts with the value returned in this ConfigEntry.
 * The list is empty if synonyms were not requested using {@link DescribeConfigsOptions#includeSynonyms
(boolean)}.
 */
public List<ConfigSynonym> synonyms() {
    return  synonyms;
}
```

Public methods of ConfigSynonym

```
public static class ConfigSynonym {

    /**
     * Returns the name of this configuration.
     */
    public String name() {
        return name;
    }

    /**
     * Returns the value of this configuration, which may be null if the configuration is sensitive.
     */
    public String value() {
        return value;
    }

    /**
     * Returns the source of this configuration.
     */
    public ConfigSource source() {
        return source;
    }
}
```

ConfigSource enum

```
public enum ConfigSource {
    DYNAMIC_TOPIC_CONFIG,           // dynamic topic config that is configured for a specific topic
    DYNAMIC_BROKER_CONFIG,          // dynamic broker config that is configured for a specific broker
    DYNAMIC_DEFAULT_BROKER_CONFIG,  // dynamic broker config that is configured as default for all brokers in
the cluster
    STATIC_BROKER_CONFIG,           // static broker config provided as broker properties at start up (e.g.
server.properties file)
    DEFAULT_CONFIG                  // built-in default configuration for configs that have a default value
}
```

Tools

kafka-configs.sh will be updated to configure defaults and overrides for dynamic configuration options for the entity type *brokers*. This will be done using the new AdminClient that talks to brokers rather than to ZooKeeper so that password encryption and config validation need to be implemented only in the broker. The full migration of ConfigCommand to use the new AdminClient will be done under KIP-248.

Two new options will be added to kafka-configs.sh under this KIP to enable broker config describe and update using the new AdminClient.

- --bootstrap-server <host>:<port>
- --command-config <config-file>

Example:

Add/update broker-specific config for broker with broker id 0:

```
bin/kafka-configs.sh  --bootstrap-server localhost:9092 --command-config /path/adminclient.props --alter --add-config 'ssl.keystore.location=keystore1.jks,ssl.keystore.password=password1' --entity-name 0 --entity-type brokers
```

Add/update cluster-wide default config:

```
bin/kafka-configs.sh  --bootstrap-server localhost:9092 --command-config /path/adminclient.props --alter --add-config 'unclean.leader.election.enable=false' --entity-default --entity-type brokers
```

Delete broker-specific config (the actual config will revert to its default value, see precedence of configs):

```
bin/kafka-configs.sh --bootstrap-server localhost:9092 --command-config /path/adminclient.props --alter --delete-config unclean.leader.election.enable --entity-type brokers --entity-name 0
```

Proposed Changes

SSL keystore

Use case: Support short-lived SSL certificates for increased security

Config scope: Broker (*/config/brokers/id*)

Config options:

- `ssl.keystore.type`
- `ssl.keystore.location`
- `ssl.keystore.password`
- `ssl.key.password`

Dynamic update changes:

- Keystore will be updated by reconfiguring the channel builder to create a new *SslFactory*. Existing connections will not be affected, new connections will use the new keystore.

Threads

Usecase:: Increase or decrease threads based on traffic pattern. This is useful for performance tuning based on metrics at runtime.

Config scope: Default for whole cluster (*/configs/brokers/<default>*)

Config options:

- `num.network.threads`
- `num.io.threads`
- `num.replica.fetchers`
- `num.recovery.threads.per.data.dir`
- `background.threads`

Dynamic update changes:

- Thread pools that don't require thread affinity will be updated to use resizable thread pools:
  - `num.io.threads`
  - `num.recovery.threads.per.data.dir`
  - `background.threads`
- `num.network.threads`: To increase threads, existing connections will continue to be processed in their processor threads and new connections will be processed on new threads. Allocation of processors for new connections will take connection count into account (instead of the current round-robin allocation) to ensure that connections are balanced across processors. When thread count is decreased, threads with the smallest number of active connections will be terminated after closing any connections being processed on those threads. New connections will be processed on remaining threads.
- `num.replica.fetchers`: Affinity of partitions to threads will be preserved for ordering.

Metrics reporters and their custom configs

Use case: Add new metrics reporter or reconfigure existing metrics reporter

Config scope: Default for whole cluster (*/configs/brokers/<default>*)

Config options:

- `metric.reporters`

Dynamic update changes:

- Addition and removal of metrics reporters will be supported. Since metrics reporters may use custom configs, dynamic configuration options for brokers may specify any custom configuration. *AdminClient* and *kafka-configs.sh* will be updated to accept custom configs.
- Metrics reporters that support reconfiguration may implement the interface *`Reconfigurable`.* If any of the configs specified in *`Reconfigurable#reconfigurableConfigs`* is updated dynamically, the reporter instance is reconfigured using *`Reconfigurable#reconfigure(Map).`*

Log Cleaner configs

Use case: Add more log cleaner threads or increase buffer size for tuning

Config scope: Default for whole cluster (*/configs/brokers/<default>*)

Config options:

- `log.cleaner.threads`
- `log.cleaner.io.max.bytes.per.second`
- `log.cleaner.dedupe.buffer.size`
- `log.cleaner.io.buffer.size`
- `log.cleaner.io.buffer.load.factor`
- `log.cleaner.backoff.ms`

Dynamic update changes:

- Config changes will be applied to existing log cleaner threads from the next iteration. Threads may be added or removed. If any of the old log cleaner threads had died due to errors, new threads will be created to replace those too so that the number of threads after an update reflects the configured `log.cleaner.threads.`

Default Topic configs

Use case: Even though topic configs can be set per-topic dynamically, it is convenient to configure properties like `unclean.leader.election.enable` consistently across the whole cluster.

Config scope: Default for whole cluster (*/configs/brokers/<default>*)

Config options (and the equivalent topic configs):

- `log.segment.bytes (segment.bytes)`
- `log.roll.ms, log.roll.hours (segment.ms)`
- `log.roll.jitter.ms, log.roll.jitter.hours (segment.jitter.ms)`
- `log.index.size.max.bytes (segment.index.bytes)`
- `log.flush.interval.messages (flush.messages)`
- `log.flush.interval.ms (flush.ms)`
- `log.retention.bytes (retention.bytes)`
- `log.retention.ms, log.retention.minutes, log.retention.hours (retention.ms)`
- `log.index.interval.bytes (index.interval.bytes)`
- `log.cleaner.delete.retention.ms (delete.retention.ms)`
- `log.cleaner.min.compaction.lag.ms (min.compaction.lag.ms)`
- `log.cleaner.min.cleanable.ratio (min.cleanable.dirty.ratio)`
- `log.cleanup.policy (cleanup.policy)`
- `log.segment.delete.delay.ms (file.delete.delay.ms)`
- `unclean.leader.election.enable (unclean.leader.election.enable)`
- `min.insync.replicas (min.insync.replicas)`
- `max.message.bytes (max.message.bytes)`
- `compression.type (compression.type)`
- `log.preallocate (preallocate)`
- `log.message.timestamp.type (message.timestamp.type)`
- `log.message.timestamp.difference.max.ms (message.timestamp.difference.max.ms)`

Dynamic update changes:

- These properties will be supported as dynamic defaults that are used by all brokers since we don't want to have different default values on different brokers. Configuration precedence for these properties will be:
  - Dynamic topic level (*/configs/topics/topicName*)
  - Dynamic broker level (*/configs/brokers/id*)
  - Dynamic default for all brokers in cluster (*/configs/brokers/<default>*)

- Static *server.properties*

Listeners and Security configs

Use cases:

- Add a new listener, e.g. to add a new security protocol or make inconsistent changes to existing security protocol (e.g change CA for SSL)
- Configure SSL/SASL configs for new security protocol
- Remove an old listener after new listener has been added to all brokers (e.g. after changing CA for SSL)
- Update advertised listeners, e.g. to set IP address after broker has been started since in some environments broker address is known only after broker has started up

Config options:

Listener Configs

- `listeners`
- `advertised.listeners`
- `listener.security.protocol.map`

Common security config

- `principal.builder.class`

SSL Configs

- `ssl.protocol`
- `ssl.provider`
- `ssl.cipher.suites`
- `ssl.enabled.protocols`
- `ssl.truststore.type`
- `ssl.truststore.location`
- `ssl.truststore.password`
- `ssl.keymanager.algorithm`
- `ssl.trustmanager.algorithm`
- `ssl.endpoint.identification.algorithm`
- `ssl.secure.random.implementation`
- `ssl.client.auth`

SASL Configs

- `sasl.jaas.config`
- `sasl.kerberos.service.name`
- `sasl.kerberos.kinit.cmd`
- `sasl.kerberos.ticket.renew.window.factor`
- `sasl.kerberos.ticket.renew.jitter`
- `sasl.kerberos.min.time.before.relogin`
- `sasl.enabled.mechanisms`
- `sasl.kerberos.principal.to.local.rules`

Dynamic update changes:

- SSL configs will be updated by reconfiguring *`ChannelBuilder`* and creating a new *`SslFactory`*. If SSL is used for inter-broker communication, inconsistent changes (e.g changing CA) should be made by adding a new listener with the new properties. This is true for SASL as well.
- SASL configuration updates will be supported using the dynamic JAAS configuration option `sasl.jaas.config`
- Updates to `advertised.listeners` will re-register the new listener in ZK. This update will be not allowed for the listener used in inter-broker communication. In addition to this, `AdminClient` will not allow updates to the listener that was used to make the alter request.
- When changes are made to listeners, additional logic will be required in the controller to broadcast the updated metadata to all brokers.
- All the security configs can be dynamically configured for new listeners. In the initial implementation, only some configs will be dynamically updatable for existing listeners (e.g. SSL keystores). Support for updating other security configs dynamically for existing listeners will be added later.

Limitations:

- Configuration updates will not be allowed for the listener used in inter-broker communication. This KIP will not allow dynamic updates to inter-broker security protocol or listener name. Support for changing inter-broker security configuration without a restart will be done in a follow-on KIP along with additional validation to ensure that all brokers have enabled the new config.

Future Work

Message Format and Inter Broker Protocol

Use case: Avoid additional rolling restarts during upgrade

Config scope: Default for whole cluster (*/configs/brokers/<default>*)

Config options:

- `log.message.format.version`
- `inter.broker.protocol.version`
- `inter.broker.listener.name`
- `security.inter.broker.protocol`
- `sasl.mechanism.inter.broker.protocol`

Dynamic update changes:

- *`AdminClient`* needs to verify that all brokers in the cluster support the new version before making an update. To enable this, the version of the broker will be added to the JSON registered by each broker during startup at */`brokers/ids/id`*. To avoid all brokers reading version information from ZooKeeper, the controller should propagate this information in `UpdateMetadataRequest`. This requires protocol change. Note also that this approach only allows verification of the version of brokers that are alive at the time `AlterConfigs` is executed.
- Additional validation is required before changing inter-broker listener or security configuration. This should ensure that the client-side and server-side configs match and also that all brokers in the cluster support the new configuration.

Compatibility, Deprecation, and Migration Plan

- Broker configuration using a properties object or the properties file *`server.properties`* will continue to be supported. So users who do not configure dynamic broker options will not be impacted.
- When downgrading from a new broker to an older version that does not support dynamic configs, any configs that were dynamically configured needs to be added to *`server.properties`*. The current configs which include dynamic configs can be obtained using *`kafka-configs.sh`*.
- If a config is renamed, broker will be updated to automatically register a dynamic config with the new name that has the value of the old config, if the old config was dynamically configured. So no action is required during upgrade. It will be up to the user to delete the old config when no downgrades are expected.

Rejected Alternatives

Deprecate static config in server.properties when a config is made dynamic

It is useful to support static configs in *`server.properties`* to enable quick start of Kafka without setting up new config options in ZooKeeper. To retain the simple user experience, we will continue to enable starting up Kafka with a properties file.