KIP-412: Extend Admin API to support dynamic application log levels

Status

Current state: *"Accepted"*

Discussion thread: here

Vote Thread: here

JIRA: KAFKA-7800

Pull Request: PR#6903

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

Note: This KIP is based on KIP-339: Create a new IncrementalAlterConfigs API

Motivation

Logging is a critical part of any system's infrastructure. It is the most direct way of observing what is happening with a system. In the case of issues, it helps us diagnose the problem quickly which in turn helps lower the MTTR.

Kafka supports application logging via the log4j library and outputs messages in various log levels (TRACE, DEBUG, INFO, WARN, ERROR). Log4j is a rich library that supports fine-grained logging configurations (e.g use INFO-level logging in `kafka.server.ReplicaManager` and use DEBUG-level in k`afka.server.KafkaApis`).
This is statically configurable through the log4j.properties file which gets read once at broker start-up.

A problem with this static configuration is that we cannot alter the log levels when a problem arises. It is severely impractical to edit a properties file and restart all brokers in order to gain visibility of a problem taking place in production.
It would be very useful if we support dynamically altering the log levels at runtime without needing to restart the Kafka process.

Log4j itself supports dynamically altering the log levels in a programmatic way and Kafka exposes a JMX API that lets you alter them. This allows users to change the log levels via a GUI (jconsole) or a CLI (jmxterm) that uses JMX.

There is one problem with changing log levels through JMX that we hope to address and that is **Ease of Use**:

- Establishing a connection - Connecting to a remote process via JMX requires configuring and exposing multiple JMX ports to the outside world. This is a burden on users, as most production deployments may stand behind layers of firewalls and have policies against opening ports. This makes opening the ports and connections in the middle of an incident even more burdensome
- Security - JMX and tools around it support authentication and authorization but it is an additional hassle to set up credentials for another system.
- Manual process - Changing the whole cluster's log level requires manually connecting to each broker. In big deployments, this is severely impractical and forces users to build tooling around it.

Proposition

Ideally, Kafka would support dynamically changing log levels and address all of the aforementioned concerns out of the box.

We propose extending the IncrementalAlterConfig/DescribeConfig Admin API with functionality for dynamically altering a single broker's log level.

This approach would also pave the way for even finer-grained logging logic (e.g log DEBUG level only for a certain topic) and would allow us to leverage the existing **AlterConfigPolicy** for custom user-defined validation of log-level changes.
These log-level changes will be **temporary** and reverted on broker restart - we will not persist them anywhere.

Public Interfaces

Protocol Changes

Users most likely need two operations for managing log levels - reading the currently-set log levels and altering them. Thus, we will add new functionality to the DescribeConfig and IncrementalAlterConfigs Admin APIs.

Resource Type

To differentiate between the normal Kafka config settings and the application's log level settings, we will introduce a new resource type - `BROKER_LOGGERS`

```
public final class ConfigResource {

    /**
     * Type of resource.
     */
    public enum Type {
        BROKER_LOGGER((byte) 8), BROKER((byte) 4), TOPIC((byte) 2), UNKNOWN((byte) 0);
    }
}
```

When `resource_type=BROKER_LOGGER`:

- we will use the existing ACL for the `Cluster` resource (as used in `IncrementalAlterConfigs/DescribeConfigs` operations).
- we will only support one of six values for the value of a config - the log levels (TRACE, DEBUG, INFO, WARN, ERROR, FATAL)

Log Level Definitions

Let's define the log levels we support and the syslog severity level (as defined in RFC-5424) they correspond to:

```
TRACE - intended to enable TRACE logs - syslog level 7 and above
DEBUG - intended to enable DEBUG logs - syslog level 7 and above
INFO - intended to enable INFO logs - syslog level 6 and above
WARN - intended to enable WARN logs - syslog level 4 and above
ERROR - intended to enable ERROR logs - syslog level 3 and above
FATAL - intended to enable FATAL logs - syslog level 0
```

To have these levels defined in code, we will create a new public class called LogLevelConfig, intended to be used by the AdminClient

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

/**
 * This class holds definitions for log level configurations related to Kafka's application logging. See KIP-
412 for additional information
 */
public class LogLevelConfig {
    /*
     * NOTE: DO NOT CHANGE EITHER CONFIG NAMES AS THESE ARE PART OF THE PUBLIC API AND CHANGE WILL BREAK USER
CODE.
     */

    /**
     * The <code>FATAL</code> level designates a very severe error
     * that will lead the Kafka broker to abort.
     */
    public static final String FATAL_LOG_LEVEL = "FATAL";

    /**
     * The <code>ERROR</code> level designates error events that
     * might still allow the broker to continue running.
     */
    public static final String ERROR_LOG_LEVEL = "ERROR";

    /**
     * The <code>WARN</code> level designates potentially harmful situations.
     */
    public static final String WARN_LOG_LEVEL = "WARN";

    /**
     * The <code>INFO</code> level designates informational messages
     *      that highlight normal Kafka events at a coarse-grained level
     */
    public static final String INFO_LOG_LEVEL = "INFO";

    /**
     * The <code>DEBUG</code> Level designates fine-grained
     * informational events that are most useful to debug Kafka
     */
    public static final String DEBUG_LOG_LEVEL = "DEBUG";

    /**
     * The <code>TRACE</code> Level designates finer-grained
     * informational events than the <code>DEBUG</code level.
     */
    public static final String TRACE_LOG_LEVEL = "TRACE";
}
```

Config Source

We will add a new `DYNAMIC_BROKER_LOGGER_CONFIG` type to the ConfigSource enum

```
public enum ConfigSource {

  DYNAMIC_BROKER_LOGGER_CONFIG, // dynamic broker logger config that is configured for a specific broker <---
NEW


  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
  UNKNOWN // source unknown e.g. in the ConfigEntry used for alter requests where source is not set
}
```

Log4jController

We will change the behavior of two getter methods.

- Log4jController#getLogLevel()
  - It would previously return "Null log level." for a logger that did not have an explicitly-configured log level
  - It will now return the ROOT logger's log level when the log level is not explicitly set for a logger
- Log4jController#getLoggers()
  - It would return all the loggers and their associated log level in the format of "logger=level". When a logger did not have an explicitly-configured log level, it would return "null" as the log level
  - It will now return the ROOT logger's log level when the log level is not explicitly set for a logger

This is clearer and more user friendly. Previously, users would need to know to search for the ROOT logger's log level to figure out what log level their desired logger was logging at.

Request/Response Overview

We will not be modifying the DescribeConfigs/IncrementalAlterConfigs request/response.
Let's go over the expected semantics when using them with the new resource type.

DescribeConfigs

```
DescribeConfigs Request (Version: 2) => [resource [config_name]] include_synonyms
    resource => resource_type resource_name
        resource_type => INT8 # BROKER_LOGGER (8)
        resource_name => STRING # ID of broker
    config_name => STRING
    include_synonyms => BOOLEAN # ignored

DescribeConfigs Response (Version: 2) => 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 # BROKER_LOGGER (8)
            resource_name => STRING # ID of broker
        configs => [config_entry synonym]
            config_entry =>
                config_name => STRING # logger name - e.g kafka.server.ReplicaManager
                config_value => STRING # log level - e.g INFO. Returns the root logger's log level if this
logger is not set explicitly yet
                read_only => BOOLEAN # false always
                config_source => INT8 # (DYNAMIC_BROKER_LOGGER_CONFIG)
                is_sensitive => BOOLEAN # false always
            synonym => # empty always
                config_name => STRING
                config_value => NULLABLE_STRING
                config_source => INT8
```

Request semantics (as defined in KIP-133) are conserved where applicable:

- an empty config_name in the request will return all existing loggers
- can be sent to any broker
- errors reported independently

IncrementalAlterConfigs

We will only support two out of the four operations for IncrementalAlterConfigs when the `resource_type=BROKER_LOGGER`.

SET: Set the log level to the desired value
REMOVE: Unsets the log level of the logger. This effectively means it is set to the root logger's log level, as the logging library goes up the chain of configured loggers until it finds one. By default - the next logger in the hierarchy is root.

```
IncrementalAlterConfigsOp => INT8
0: SET
1: REMOVE # sets log level to the root logger's level
2: APPEND # NOT SUPPORTED
3: SUBTRACT # NOT SUPPORTED

IncrementalAlterConfigsRequest (Version: 0) => [resources] validate_only
 validate_only => BOOLEAN
 resources => resource_type resource_name [configs]
    resource_type => INT8 # BROKER_LOGGER (8)
    resource_name => STRING # ID of broker
    configs => config_name config_op config_value
        config_name => STRING
        config_op => INT8 # support SET and APPEND only
        config_value => NULLABLE_STRING

IncrementalAlterConfigsResponse (Version: 0) => [responses]
  responses => resource_type resource_name error_code error_message
  resource_type => INT8 # BROKER_LOGGER (8)
  resource_name => STRING # ID of broker
  error_code => INT16
  error_message => NULLABLE_STRING
```

Request semantics (as defined in KIP-133 and KIP-339) are conserved where applicable:

- validate_only mode does not alter the log level
- can be sent to any broker
- non-transactional
- if config key is duplicated, INVALID_REQUEST gets returned for all those duplicate keys
- INVALID_REQUEST will always be returned for APPEND/SUBTRACT operations

Error Handling

In the case of an invalid config_value or an invalid/non-existent logger name, the broker will return an `INVALID_CONFIG` (40) error for that config resource (BROKER_LOGGER).
INVALID_REQUEST will be returned when attempting to unset the ROOT logger's log level

Tools Changes

`kafka-configs.sh` will be extended to support the new resource type via `--entity-type broker-logger`.

Examples:

```
bin/kafka-configs.sh --bootstrap-server localhost:9092 --describe --entity-type broker-loggers --entity-name 0 //
show all the log levels for broker 0

bin/kafka-configs.sh --bootstrap-server localhost:9092 --alter --add-config "kafka.server.ReplicaManager=WARN,
kafka.server.KafkaApis=DEBUG" --entity-type broker-loggers --entity-name 0 // set some log levels for broker 0

bin/kafka-configs.sh --bootstrap-server localhost:9092 --alter --delete-config kafka.server.ReplicaManager --
entity-type broker-loggers --entity-name 0 // will set the log level to the ROOT logger level
```

Compatibility, Deprecation, and Migration Plan

Compatibility

Since we are only adding new functionality under a new resource type, this KIP should not have compatibility issues with older versions.

Kafka will continue to expose the JMX API for configuring log levels. The only difference is that it will not return "null" for unset loggers now but rather return the root logger's log level.

Since we want to deprecate `AlterConfigs`, that API will **not** support altering log levels.

Testing

We should be able to create a JUnit integration test inside AK that can call the Admin API methods to modify the log-level and have access to Log4j in order to verify that the levels are changed.

Migration Plan

`AlterConfigPolicy` implementations will need to be updated to account for the new config type.

Rejected Alternatives

1. Extend Kafka's JmxTool class to support changing log levels dynamically
    a. Does not properly address the concerns outlined in the Motivation section
        i. Can abstract away a bit of the complexity but still has the fundamental flaws of no security and not being easy to use
2. Create new Admin API commands for reading and altering log levels
    a. Will result in more boilerplate and some code duplication.
    b. Will result in seemingly needless Admin API command bloat
    c. Altering log levels can be seen as a form of altering configurations and seems intuitive use the same API
    d. Will require separate Policy class
3. Add new `config_type` field in Alter/Describe request/responses or new map of `logger=>log_level`
    a. Results in more code and field bloat in request/response without clear benefits of extendability
4. Support turning log levels to OFF
    a. Log4j supports a log level called OFF. We decided to not expose this as there is never a compelling reason to turn off application logs.