KIP-676: Respect logging hierarchy

Status

Current state: *Approved*

Discussion thread: *here*

JIRA: *KAFKA-10469*

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

Motivation

Log4j (and log4j2, see KIP-653) use a hierarchical model for configuring loggers within an application. A logger is an entity which can generate logging. Loggers have names which are usually based on the fully-qualified class name of the class where the logger is used. Thus dots separate the components of a logger name.
One logger is an ancestor of another if the components in its name are a (strict) prefix of the components of the other's name. Such ancestor loggers are not typically used to emit logging messages, but exist to configure descendent loggers in a hierarchical way. There is a root logger (which in log4j has an empty name, "", but in Kafka is identified by the name "root"), which is the ancestor of all other loggers. If follows that loggers which are not explicitly configured, and with no intermediate ancestor which _is_ configured, will inherit their configuration from the root logger.

Loggers may (but do not have to be) be configured in a configuration file. When a logger is not explicitly configured it inherits its configuration from the first ancestor logger which *is* configured.

Kafka exposes a number of APIs for describing and changing logger levels:

- The Kafka broker exposes the `DescribeConfigs` RPC with the `BROKER_LOGGER` config resource.
- Broker logger levels can also be configured using the `Log4jControllerMBean` MBean, exposed through JMX as `kafka:type=kafka.Log4jController`.
- Kafka Connect exposes the `/admin/loggers` REST API endpoint for describing and changing logger levels.

When accessing a logger's level the first two of these APIs do not respect the logger hierarchy.

Instead, if the logger's level is not explicitly set the level of the root logger is used, even when an intermediate logger is configured with a different level.

This creates unexpected and incorrect behaviour because the level, as obtained through a Kafka API, can report that the level of a given logger that is actually inheriting its configuration from an ancestor has the level of the root logger. For example, if the root logger's level is `INFO` and `kafka.foo` is configured as `TRACE` then `kafka.foo.bar` (which is not explicitly configured) is actually logging at `TRACE` (inherited from `kafka.foo`), but will be incorrectly reported to be logging at `INFO`.

The current behaviour for `DescribeConfigs` is explicitly defined in at least KIP-412 (search for "Returns the root logger's log level if this logger is not set explicitly yet").

Changing this amounts to changing the behavior of this public API.

Public Interfaces

No changes except to the behaviour of `DescribeConfigs` RPC the `/admin/loggers` endpoint in Kafka connect and the `Log4jControllerMBean` MBean.

Proposed Changes

Describe the new thing you want to do in appropriate detail. This may be fairly extensive and have large subsections of its own. Or it may be a few sentences. Use judgement based on the scope of the change.

Compatibility, Deprecation, and Migration Plan

This change would impact any clients which were be working around the current buggy behaviour.

Reusing the previous example, after broker was upgraded to a version with this KIP implemented, `kafka.foo.bar` would appear to change from `INFO` to `TRACE` "on its own", without a user having explicitly reconfigured it. However, the level at which it was actually logging would remain `TRACE` both before and after upgrade.

If this KIP were accepted for Kafka 3.0 then it would be less surprising to those clients.

Rejected Alternatives

None identified.