KIP-606: Add Metadata Context to MetricsReporter

Status

Current state: *Adopted*

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

Metric Reporter plugins have the ability to collect metrics from various Kafka components (broker, connect) and Kafka client libraries (producer, consumer, admin client, streams) and expose/send those metrics to other systems.

However, the current reporter interface does not give much context about the system exposing those metrics. For instance, it is currently difficult to infer which component or library is exposing those metrics: a broker, a connect worker, or a client library running within any of those components.

There are two issues we would like to address:

1. Letting metric reporters know which component is exposing the metrics

The stock metrics reporter implementation – `JmxReporter` already gets passed some additional context via the JMX prefix (e.g. `kafka.server` `kafka.streams`, `kafa.admin.client`, etc.) at instantiation. However, this information is not available to additional metrics reporters configured at startup.

2. Exposing the context in which those metrics are collected

For client metrics, it is not straightforward for metrics reporter plugins to know which service the client is run in, e.g. for a connect sink task, the metrics reporter configured in the Kafka consumer has no information about connect worker in which it runs.

This makes it impractical to use metrics reporter plugins to correlate all the metrics exposed by a service.

For Kafka brokers, we currently resort to various workarounds to get the broker id (the config is manipulated before passed to the metrics reporter) or cluster id (only accessible via ClusterResourceListener).

Some of that information is valuable to expose in metrics and it would be useful to have a consistent mechanism to provide this context to metrics reporters.

To address those issues, we propose to add a mechanism to pass additional metadata to metrics reporters, so they can properly infer the context in which metrics are registered and which component exposes them.

We also propose to add new client configurations, so any application can propagate metrics context metadata from the parent application down to the metric reporter instantiated by the client.

Public Interfaces

MetricsReporter interface changes

To expose additional metadata fields to metric reporter plugins, we propose to add a new callback method to the MetricsReporter interface

```
public interface MetricsReporter extends Reconfigurable, AutoCloseable {
    [...]

    /**
     * Provides context metadata for the service or library exposing metrics
     *
     * @param metricsContext the metric context
     */
    @InterfaceStability.Evolving
    default void contextChange(MetricsContext metricsContext) {}
}
```

The MetricsContext will encapsulate this additional metadata as a map of key-value pairs.

```
/**
 * MetricsContext encapsulates additional metadata about metrics exposed via a
 * {@link org.apache.kafka.common.metrics.MetricsReporter}
 *
 * The metadata map provides following information:
 * - a <code>_namespace</node> field indicating the component exposing metrics
 *   e.g. kafka.server, kafka.consumer
 *   {@link JmxReporter} uses this as prefix for mbean names
 *
 * - for clients and streams libraries: any freeform fields passed in via
 *   client properties in the form of `metrics.context.<key>=<value>
 *
 * - for kafka brokers: kafka.broker.id, kafka.cluster.id
 * - for connect workers: connect.kafka.cluster.id, connect.group.id
 */
@InterfaceStability.Evolving
public interface MetricsContext {

    /* predefined fields */
    String NAMESPACE = "_namespace"; // metrics namespace, formerly jmx prefix

    /**
     * Returns metadata fields
     */
    public Map<String, String> contextLabels();
}
```

Connect, Streams and Client configuration changes

A new configuration prefix will be introduced for connect, streams and client configs, so applications can pass context metadata to embedded client libraries.

`metrics.context.<key>=<value>`

Connect and streams will pass on their context configuration to the clients they instantiate. Similar to other client configurations, explicit client config overrides would take precedence over values passed through from the main config.

The configured key value pairs are then passed by the clients to the configured metrics reporters via the MetricsContext.

For instance, configuring a client with `metrics.context.foo.bar=baz` would add the field `foo.bar` mapped to the value `baz` in the MetricsContext metadata the client metrics reporter.

JmxReporter changes

The `JmxReporter(String prefix)` constructor will be deprecated in favor of the default constructor.

The prefix will be passed to the reporter via the metrics context instead. An internal metric context field `_namespace` will hold the existing jmx prefix values.

By convention, pre-defined metric context fields that are guaranteed to be present would be prefixed with an underscore.

Proposed Changes

Remove the special status of JmxReporter

Existing usages of `JmxReporter(String prefix)` will be removed and replaced with the default constructor, in line with what we do for metric reporter plugins defined via configuration.

The JmxReporter will give precedence to the `_namespace` field from the `MetricsContext` as JMX prefix over the `prefix` field passed in via the deprecated constructor.

All components and libraries currently instantiating metrics reporters will define the necessary metric context and add the `_namespace` field to the metrics context.

Client and Streams libraries

In addition to the above, clients and streams libraries will read in additional `metrics.context.<field>=<value>` entries passed in via configuration and add them to the metric context before calling `contextChange(MetricsContext metricsContext)`.

Kafka Brokers

Kafka brokers will pass in fully resolved fields `kafka.broker.id` and `kafka.cluster.id` to the metric context.

In addition, similar to client, any `metrics.context.<field>=<value>` entries passed in via broker configuration will also be added to the metrics context.

Connect Workers

Connect workers will pass the following fields to the metric context:

- `connect.kafka.cluster.id` to indicate the backing Kafka cluster
- `connect.group.id` to indicate the group id used for worker coordination (only in distributed mode)

Connect will also propagate these fields to embedded clients via the `metrics.context.` client properties.

Compatibility, Deprecation, and Migration Plan

`contextChange(MetricsContext metricsContext)` will be added as a default method to `MetricsReporter` to support backward compatibility with existing implementations. No changes are required for existing metric reporters since we are not changing any other behavior.

`metrics.context.` configuration properties are optional, and as such would not impact any upgrades.

Since the properties are namespaced, they are unlikely to conflict with configs from other extensions / plugins.

Rejected Alternatives

- **Add additional metadata to the tags defined in** `MetricsConfig`
  `MetricsConfig` tags are tightly coupled to `MetricNameTemplate` objects, and require metric names to provide placeholders for all the tags, which make it difficult to add new tags without changing the metric names.
- **Add additional metadata to the ClusterResource class**
  Metrics reporters could already get some Kafka cluster metadata information (e.g, Kafka cluster id) by implementing the ClusterResourceListener interface introduced in KIP-78. We considered adding additional context metadata to this class, but ultimately rejected that idea. ClusterResource is tightly coupled to a Kafka cluster. Adding metadata of other components to ClusterResource would be ill-suited and cause for confusion. The `ClusterResourceListener.onUpdate(ClusterResource)` method also has well-defined semantics for broker and client metadata updates, which would make implementing MetricsReporters more difficult.
- **Pass in additional context via the Configurable/Reconfigurable interface**
  As noted, some context metadata such as `broker.id` is currently available via configuration, but relies on workarounds to manipulate the config passed to metrics reporters when they are defined dynamically. While it might be possible to use `reconfigure(...)` instead to pass context values when they are known, this would require metric reporters to know all possible context keys in advance to return in `reconfigurableConfigs`. As a result, one would have to make code changes to reporter plugins with every release just to support new fields added to the context. That would be akin to asking metrics reporter to know what metric tags are exposed in advance, which seemed convoluted.