KIP-982: Access SslPrincipalMapper and kerberosShortNamer in Custom KafkaPrincipalBuilder

- Status
- Introduction
- Motivation
- Proposed Changes
- Compatibility, Deprecation, and Migration Plan
- Rejected Alternatives
  - Alternative-1: Changes in ChannelBuilders
  - Alternative-2: Changes in SslPrincipalMapper
  - Alternative-3: Changes in SslChannelBuilder
- Test Plan
- Documentation Updates

Status

Current state: *Under Discussion*

Discussion thread: *here*

***Pull Request thread**: here & Pull Request*

***Vote 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).

Introduction

KAFKA-15452 reports a bug that affects the ability of custom KafkaPrincipalBuilder implementations to access SslPrincipalMapper and kerberosShortNamer. This limits the ability to parse Regex Rules from BrokerSecurityConfigs SSL_PRINCIPAL_MAPPING_RULES_CONFIG, resulting in a lack of support for Mapping Rules as SslPrincipalMapper is null. This KIP proposes a solution to address this issue at the ChannelBuilders or SslChannelBuilder level.

Motivation

The main motivation behind this KIP is to enhance the flexibility and usability of custom KafkaPrincipalBuilder implementations by allowing them to access SslPrincipalMapper and kerberosShortNamer. This will enable support for Mapping Rules and improve the overall security configuration of Kafka brokers.

Proposed Changes

The proposed changes aim to enhance the flexibility and usability of **custom KafkaPrincipalBuilder** implementations by introducing interfaces and subclasses to handle different types of principals and their configurations.

Specifically, two new interfaces have been introduced: **KerberosPrincipalBuilder** and **SSLPrincipalBuilder**, both of which extend the existing **KafkaPrincipalBuilder** interface. These interfaces provide methods to set the necessary configurations for building Kafka principals with Kerberos authentication and SSL authentication, respectively.

Additionally, the **ChannelBuilders** class has been updated to accommodate these changes and provide support for custom principal builders with different authentication mechanisms.

Firstly, the **KerberosPrincipalBuilder** interface is defined as follows:

```
public interface KerberosPrincipalBuilder extends KafkaPrincipalBuilder {
    void buildKerberosPrincipalBuilder(KerberosShortNamer kerberosShortNamer);
}
```

Secondly, the **SSLPrincipalBuilder** interface is defined as follows:

```
public interface SSLPrincipalBuilder extends KafkaPrincipalBuilder {
    void buildSSLPrincipalBuilder(SslPrincipalMapper sslPrincipalMapper);
}
```

In the **ChannelBuilders** class, the **createPrincipalBuilder** method has been updated to support custom principal builders with different authentication mechanisms. When creating a custom principal builder, the appropriate interface methods are called to set the necessary configurations.

Here is the updated code snippet:

```
public static KafkaPrincipalBuilder createPrincipalBuilder(Map<String, ?> configs,
                                                           TransportLayer transportLayer,
                                                           Authenticator authenticator,
                                                           KerberosShortNamer kerberosShortNamer,
                                                           SslPrincipalMapper sslPrincipalMapper) {
        Class<?> principalBuilderClass = (Class<?>) configs.get(BrokerSecurityConfigs.
PRINCIPAL_BUILDER_CLASS_CONFIG);
        final KafkaPrincipalBuilder builder;

        if (principalBuilderClass == null || principalBuilderClass == DefaultKafkaPrincipalBuilder.class) {
            builder = new DefaultKafkaPrincipalBuilder(kerberosShortNamer, sslPrincipalMapper);
        } else if (SSLPrincipalBuilder.class.isAssignableFrom(principalBuilderClass)) {
            SSLPrincipalBuilder sslPrincipalBuilder = (SSLPrincipalBuilder) Utils.newInstance
(principalBuilderClass);
            sslPrincipalBuilder.buildSSLPrincipalBuilder(sslPrincipalMapper);
            builder = sslPrincipalBuilder;
        } else if (KerberosPrincipalBuilder.class.isAssignableFrom(principalBuilderClass)) {
            KerberosPrincipalBuilder kerberosPrincipalBuilder = (KerberosPrincipalBuilder) Utils.newInstance
(principalBuilderClass);
            kerberosPrincipalBuilder.buildKerberosPrincipalBuilder(kerberosShortNamer);
            builder = kerberosPrincipalBuilder;
        } else if (KafkaPrincipalBuilder.class.isAssignableFrom(principalBuilderClass)) {
            builder = (KafkaPrincipalBuilder) Utils.newInstance(principalBuilderClass);
        } else if (org.apache.kafka.common.security.auth.PrincipalBuilder.class.isAssignableFrom
(principalBuilderClass)) {
            org.apache.kafka.common.security.auth.PrincipalBuilder oldPrincipalBuilder =
                    createPrincipalBuilder(principalBuilderClass, configs);
            builder = DefaultKafkaPrincipalBuilder.fromOldPrincipalBuilder(authenticator, transportLayer,
                    oldPrincipalBuilder, kerberosShortNamer);
        } else {
            throw new InvalidConfigurationException("Type " + principalBuilderClass.getName() + " is not " +
                    "an instance of " + org.apache.kafka.common.security.auth.PrincipalBuilder.class.getName()
+ " or " +
                    KafkaPrincipalBuilder.class.getName());
        }

        if (builder instanceof Configurable)
            ((Configurable) builder).configure(configs);

        return builder;
    }
```

These changes ensure that custom **KafkaPrincipalBuilder** implementations can now access **SslPrincipalMapper** and **KerberosShortNamer**, enabling support for **mapping rules** and improving the overall security configuration of Kafka brokers.

The introduced interfaces provide a clear and extensible way to handle different types of principals, making the solution more modular and maintainable.

Compatibility, Deprecation, and Migration Plan

The proposed changes are fully backward-compatible and do not require any deprecations or migration efforts. Existing custom KafkaPrincipalBuilder implementations will continue to work without any modifications. However, they can be updated to utilize the new functionality if desired.

Rejected Alternatives

Alternative-1: Changes in ChannelBuilders

In the **ChannelBuilders** class, we will update the existing **createPrincipalBuilder** method to Check for an additional Constructor:

```
public static KafkaPrincipalBuilder createPrincipalBuilder(Map<String, ?> configs,
                                                           TransportLayer transportLayer,
                                                           Authenticator authenticator,
                                                           KerberosShortNamer kerberosShortNamer,
                                                           SslPrincipalMapper sslPrincipalMapper) {
        Class<?> principalBuilderClass = (Class<?>) configs.get(BrokerSecurityConfigs.
PRINCIPAL_BUILDER_CLASS_CONFIG);
        KafkaPrincipalBuilder builder;

        if (principalBuilderClass == null || principalBuilderClass == DefaultKafkaPrincipalBuilder.class) {
            builder = new DefaultKafkaPrincipalBuilder(kerberosShortNamer, sslPrincipalMapper);
        } else if (KafkaPrincipalBuilder.class.isAssignableFrom(principalBuilderClass)) {
            try {
                Constructor<?> constructor = principalBuilderClass.getConstructor(KerberosShortNamer.class,
SslPrincipalMapper.class);
                builder = (KafkaPrincipalBuilder) constructor.newInstance(kerberosShortNamer,
sslPrincipalMapper);
            } catch (NoSuchMethodException e) {
                builder = (KafkaPrincipalBuilder) Utils.newInstance(principalBuilderClass);
            } catch (InstantiationException | IllegalAccessException | InvocationTargetException e) {
                throw new RuntimeException("Error instantiating custom KafkaPrincipalBuilder", e);
            }
        } else if (org.apache.kafka.common.security.auth.PrincipalBuilder.class.isAssignableFrom
(principalBuilderClass)) {
            org.apache.kafka.common.security.auth.PrincipalBuilder oldPrincipalBuilder =
                    createPrincipalBuilder(principalBuilderClass, configs);
            builder = DefaultKafkaPrincipalBuilder.fromOldPrincipalBuilder(authenticator, transportLayer,
                    oldPrincipalBuilder, kerberosShortNamer);
        } else {
            throw new InvalidConfigurationException("Type " + principalBuilderClass.getName() + " is not " +
                    "an instance of " + org.apache.kafka.common.security.auth.PrincipalBuilder.class.getName()
+ " or " +
                    KafkaPrincipalBuilder.class.getName());
        }

        if (builder instanceof Configurable)
            ((Configurable) builder).configure(configs);

        return builder;
    }
```

In this will pass the custom KafkaPrincipalBuilder with SslPrincipalMapper and kerberosShortNamer objects if Constructor is available.

This approach is solving a problem for a certain Principal type and not all of the principals have to use KerberosShortNamer, it is related if the credentials are provided by the Kerberos authentication.

Alternative-2: Changes in SslPrincipalMapper

An alternative workaround is to read the configuration and build another SslPrincipalMapper in the custom KafkaPrincipalBuilder implementation. However, this approach is less efficient and may lead to code duplication. The proposed solution provides a more elegant and efficient way to address the issue.

Alternative-3: Changes in SslChannelBuilder

In the **SslChannelBuilder** class, we will add a new method **configurePrincipalBuilder** that configures the custom KafkaPrincipalBuilder with SslPrincipalMapper and kerberosShortNamer:

```
protected void configurePrincipalBuilder(Map<String, ?> configs) {
    // implementation
}
```

This method will be called in the **ChannelBuilders** class to configure the custom KafkaPrincipalBuilder.

Test Plan

The test plan for this KIP includes:

1. Unit tests to ensure that the new methods in **ChannelBuilders** work as expected.
2. Integration tests to ensure that custom KafkaPrincipalBuilder implementations can access SslPrincipalMapper and kerberosShortNamer properly.
3. Performance tests to ensure that the proposed changes do not introduce any performance regressions.

Documentation Updates

The official Kafka documentation will be updated to reflect the new methods and their usage in custom KafkaPrincipalBuilder implementations. This includes updates to the JavaDocs and any relevant sections in the Kafka Security documentation.