KIP-383: Pluggable interface for SSL Factory

Status

Current state: Discarded, replaced by KIP-519: Make SSL context/engine configuration extensible

Discussion thread: *here*

JIRA: *here*

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

Motivation

It has been a common request to be able to customize SslFactory beyond the existing configuration options. For example, users want encrypted passwords, an alias to choose a certificate, OCSP support, etc. For these, alternative config options make the most sense because these are generic config options that are useful in many environments, so we should support these without having to write Java code.

There are situations where adding configuration options will never be able to satisfy the requirement. For example, the user may need to reuse an existing SSL implementation or share an implementation across multiple applications. For those cases, the simplest approach would be to support the customization of the SslFactory implementation.

A custom SslFactory implementation might also serve as a stop gap measure when a user needs an SSL feature immediately but the necessary configuration option is not yet available.

Of course it would be easy to simply change the Kafka source code and ship a customized distribution. The idea is you should be able to replace the implementation through configuration, without rebuilding a custom Kafka distribution or resorting to classpath tricks to shadow Kafka classes.

SslFactory handles two different tasks: creating the SSLContext and reconfiguration. There is no need to make reconfiguration pluggable. This functionality should remain in SslFactory to be shared by all pluggable implementations. SslFactory only needs to delegate the creation of the SSLContext to the pluggable interface.

Since SslFactory will handle reconfiguration, the idea is to make the configuration immutable in the pluggable factory. SslFactory would create a new pluggable factory every time the configuration changes. The pluggable factory creates its SSLContext when it is configured and never changes it. It turns out SslFactory does not really need the SSLContext, so it can use the new pluggable factory as an SSLEngine factory instead.

Public Interfaces

This KIP will introduce a new configuration option ssl.sslengine.factory.class to be added to SslConfigs. This will automatically add the new option to AdminClientConfig, ConsumerConfig and ProducerConfig.

A new public interface named SslEngineFactory will be created. A default implementation named DefaultSslEngineFactory with the existing behavior will be included with Kafka. The implementation is private, but the class name will leak as the default value of ssl.sslengine.factory.class

An application developer can implement SslEngineFactory and set ssl.sslengine.factory.class to the fully qualified class name to instruct Kafka to use this implementation instead of the default implementation.

Proposed Changes

A new configuration option will be added to SslConfigs:

```
  public static final String SSL_SSLENGINE_FACTORY_CLASS_CONFIG = "ssl.sslengine.factory.class";
  public static final String SSL_SSLENGINE_FACTORY_CLASS_DOC = "...";
  public static final String DEFAULT_SSL_SSLENGINE_FACTORY_CLASS = "org.apache.kafka.common.security.ssl.
DefaultSslEngineFactory";

  config.define(SslConfigs.SSL_SSLENGINE_FACTORY_CLASS_CONFIG, ConfigDef.Type.CLASS, SslConfigs.
DEFAULT_SSL_SSLENGINE_FACTORY_CLASS, ConfigDef.Importance.LOW, SslConfigs.SSL_SSLENGINE_FACTORY_CLASS_DOC)
```

A new public interface named SslEngineFactory will be created.

```
  public interface SslEngineFactory {

      void configure(Map<String, ?> configs, Mode mode);

      SSLEngine createSslEngine(String peerHost, int peerPort);
```

```
}
```

SslFactory finds the class name of the SslEngineFactory implementation in the ssl.sslengine.factory.class config and instantiates it using the default constructor by calling Class.newInstance(). The configs and mode are then passed to configure. This creates the SSLContext which remains private to the SslEngineFactory implementation. SslFactory can now call createSslEngine(peerHost, peerPort) as often as needed.

This assumes SslFactory copies the configs and overwrites ssl.client.auth when there is a clientAuthConfigOverride.

When reconfiguring, only the keystore and truststore configs are allowed to change. This is the existing behavior and this KIP simply preserves it. SslFactory must keep a copy of the previous configs to know the original values for the other configs. This copy also guarantees we don't overwrite anything in the caller's configs.

SslFactory must inspect the keystore and truststore to detect a future change in configuration. In particular, it needs to keep the lastModified times, plus the CertificateEntries of the keystore. The configuration might change during the small interval between the inspection and loading the files in the SslEngineFactory implementation. By always inspecting the files before calling configure, we make sure the SslEngineFactory implementation will always have the same or newer configuration and reconfiguration will never miss an update later.

It is allowed to keep ssl.keystore.location and/or ssl.truststore.location empty. This means the discovery of the keystore and truststore is fully delegated to the SslEngineFactory implementation. This turns off reconfiguration when both the old and new configuration delegates to the SslEngineFactory implementation.

SslFactory.sslContext() will be removed because the SslContext is now private to the SslEngineFactory implementation.

Compatibility, Deprecation, and Migration Plan

This KIP is considered backwards compatible. SslFactory is not part of the Kafka public API. The SslFactory API remains the same anyway except for the removal of sslContext().

EchoServer is the only caller of SslFactory.sslContext(), This is an internal test so backwards compatibility does not apply. We will update EchoServer to call DefaultSslFactory directly instead.

The default value for the new config ssl.sslengine.factory.class is org.apache.kafka.common.security.ssl.DefaultSslEngineFactory which implements the existing behavior.

Rejected Alternatives

Kafka could define a new configuration option to hold an instance of SSLSocketFactory. This is similar to many Java libraries that accept an instance of SSLSocketFactory. This was rejected because Kafka tries to be language neutral. It was thought it would make it more difficult to support C and Python.

Kafka's naming convention does not use a special tag for interfaces. Accordingly, these interface names were rejected ISslEngineFactory, SslEngineFactoryIFace, SslEngineFactoryInterface.

We could expose the SSLContext with an accessor in the SslEngineFactory interface, but it was only used by an internal test, so we decided to leave it out.

Kafka is not consistent to name configs that hold class names. Compare [partitioner.class, interceptor.classes, principal.builder.class, sasl.server.callback.handler.class, sasl.client.callback.handler.class, sasl.login.class] versus [key.deserializer, value.deserializer, key.serializer, value.serializer]. It appears the serializer/deserializer configs are a special case. Therefore the name ssl.sslengine.factory.class was selected instead of ssl.sslengine.factory

We could add clientAuthConfigOverride as a third argument for SslEngineFactory.configure(). This looked like an internal feature of Kafka that leaked into the more general interface. We thought overwriting ssl.client.auth in a copy of the configs was cleaner.

We could pass the keystore and truststore as arguments to the SslEngineFactory since SslFactory already loaded them. This felt awkward since the keystore and truststore configs would still be present in the configs passed to configure(). It also felt clumsy when the SslEngineFactory uses its own keystore and truststore, for example, when reusing an SSLContext from the rest of the application.

We need the ability to send custom configuration options to the PluggableSslFactory implementation. This could be covered in another KIP.