SPIP: Support Kafka delegation token in Structured Streaming

Author: Gabor Somogyi

Background and Motivation

In Spark 2.4 Structured Streaming can connect to Kafka in a secure way only by using KDC and no delegation token support available. Until now one of the possible solutions was to distribute keytab files to executors (please see this example application for further details) but there are a few commonly raised concerns with this approach:

- It's not considered the best security practice to ship keytabs around.
- In case of a large number of Kafka topic partitions, all executors may try logging in to the KDC at the same time, potentially leading to DDOS attack.

Kafka added delegation token support in version 1.1.0 (further information can be found here) and Spark updated it's Kafka client library to 2.0.0 in SPARK-18057. These events made delegation token support available in Spark.

As I haven't found example application for Kafka's delegation token I've created a standalone one.

This document proposes a solution to have Kafka delegation token support in Structured Streaming.

Target Personas

Everybody using Structured Streaming with secured Kafka.

Goals

- Add Kafka delegation token support for Structured Streaming.

Non-Goals

- Add Kafka delegation token support for DStreams.

Proposed API Changes

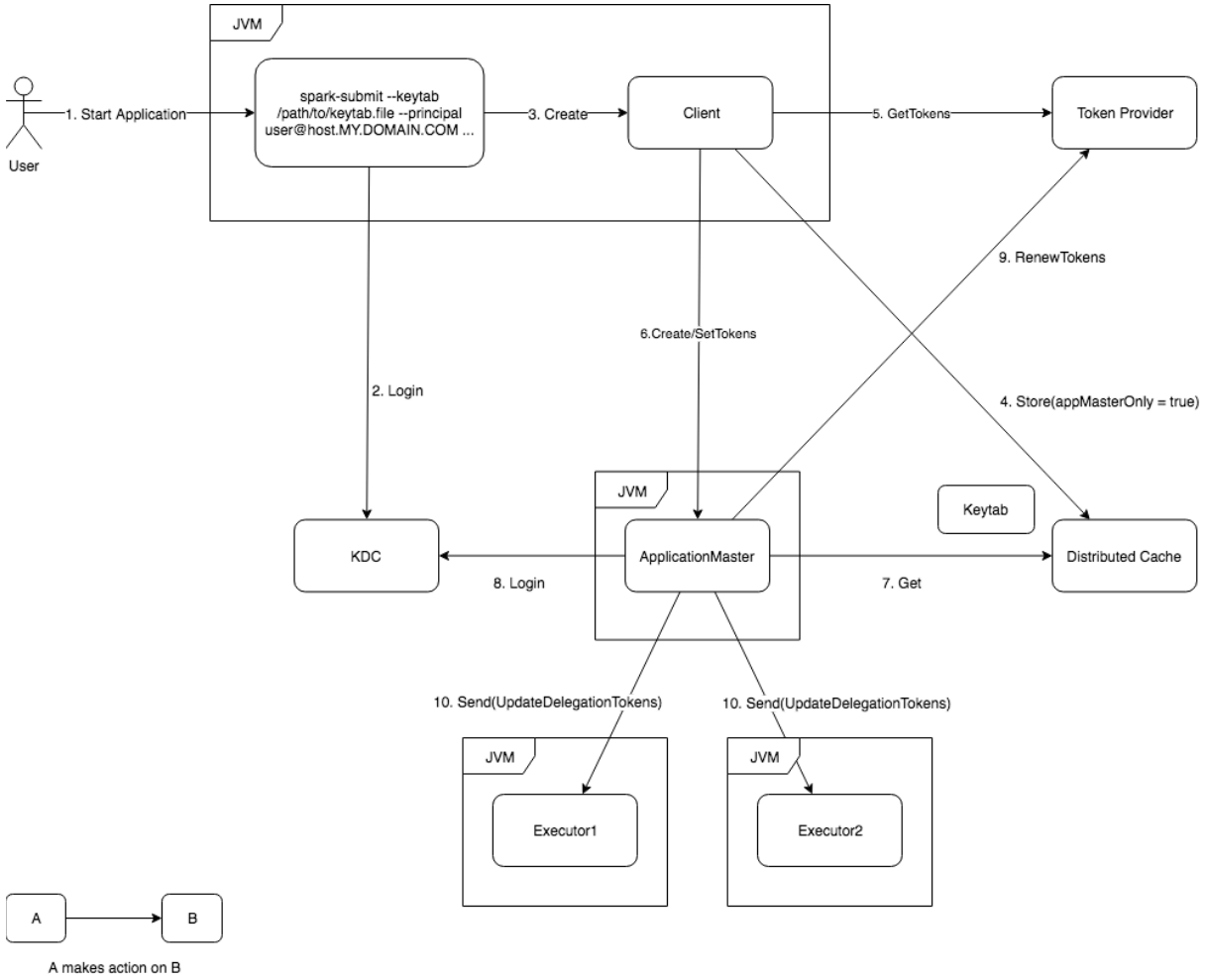
Additional parameters has to be provided in Spark's configuration but only if delegation token needed. Please see Design Sketch.

Design Sketch

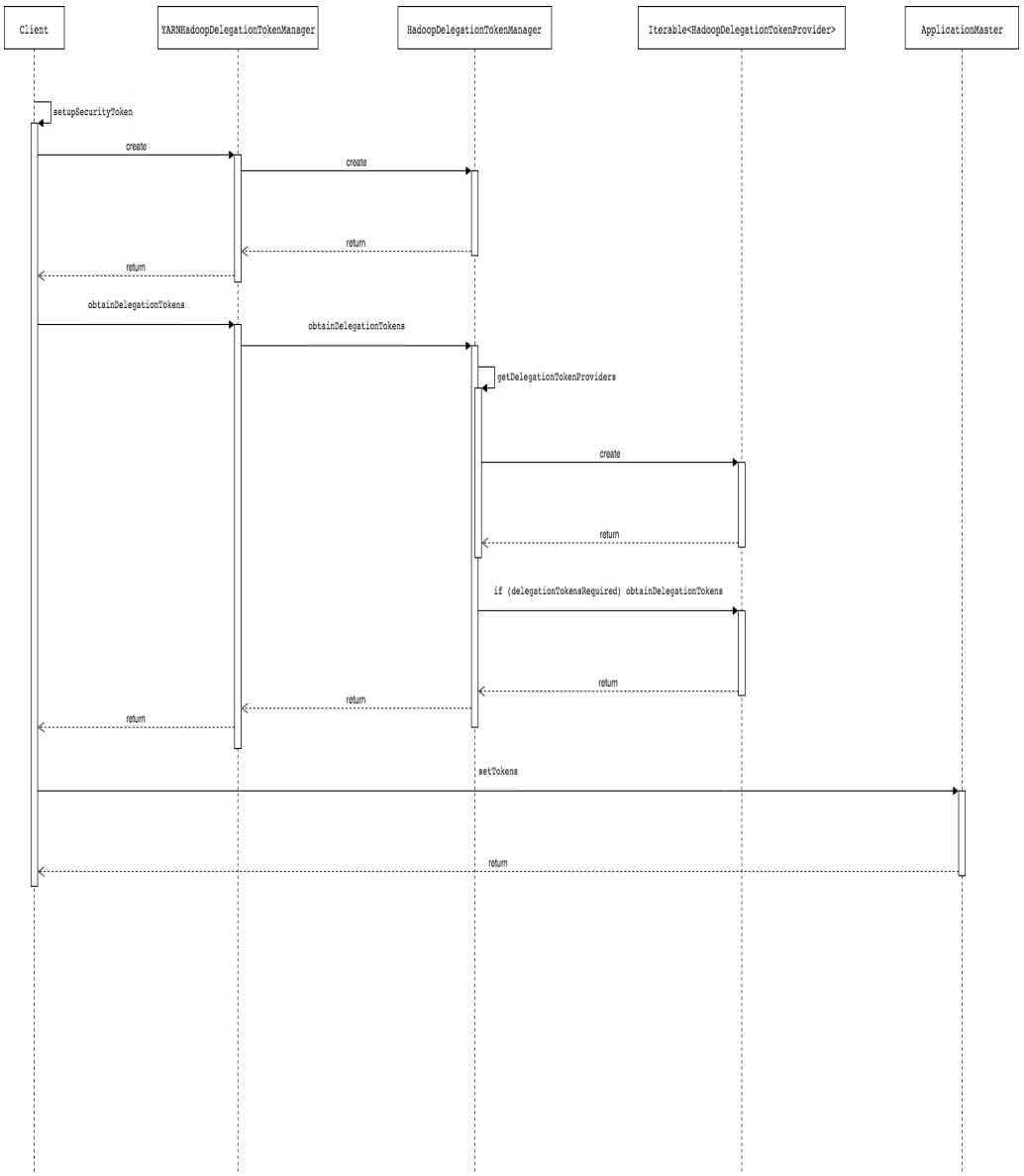
In Spark there is already a framework to handle delegation tokens for HDFS, HBase and Hive. In this proposal this infrastructure intended be extended.

Please note the following diagrams are not covering all use-cases and not always reflecting exact class/function names, these are just for presentation purposes.

High level diagram about the existing framework in Yarn cluster mode with keytab file:



And here is a sequence diagram how tokens obtained:



In case of mesos instead of **YARNHadoopDelegationTokenManager** **MesosHadoopDelegationTokenManager** has similar functionality.

As it's shown on the sequence diagram **HadoopDelegationTokenManager** has an **Iterable<HadoopDelegationTokenProvider>**. Each provider represents a service.
The proposal is to add a Kafka specific delegation token provider similar to **HBaseDelegationTokenProvider**. This will load a Kafka utility class through reflection which will be placed in external/kafka-0-10-sql not to pollute core projects with Kafka specific details. The utility class will contain all the Kafka related specifics to get the token. The planned interface is something like:

def obtainToken(sparkConf: SparkConf): (Token[_ <: TokenIdentifier], Long) = ...

Initial token fetching is at the very beginning in the Spark process, before parsing source/sink parameters. Passing Kafka parameters now possible on source/sink (see an example here) which is late from delegation token point of view.

The proposal is to define parameters in Spark's config which makes them available through **SparkConf**. These parameters would be required only if delegation token was used.

In order to turn off provider the following parameter can be used:

- spark.security.credentials.kafka.enabled: Optional, Default: true

In order to get/provide the token itself the following Kafka parameters needed (further information about these parameters can be found here):

- bootstrap.servers: Optional, Default: null
- security.protocol: Optional, Default: SASL_SSL
- sasl.kerberos.service.name: Optional, Default: kafka
- ssl.truststore.location: Optional, Default: null
- ssl.truststore.password: Optional, Default: null
- ssl.keystore.location: Optional, Default: null
- ssl.keystore.password: Optional, Default: null
- ssl.key.password: Optional, Default: null

The proposal is to put them into Spark's config with "spark.kafka." prefix with the following rules.

- If the parameters are not provided the already existing Kafka source/sink parameter handling applies.
- If the parameters provided they will be used the following ways:
    - Check if token required
    - Get/renew token

The proposed condition to decide whether token has to be obtained:
(bootstrap.servers defined && security.protocol is in (SASL_SSL, SSL, SASL_PLAINTEXT))

Then the obtained delegation token will be sent through the already existing **UpdateDelegationTokens** event to the executors.

Renewal is triggered by the already existing **AMCredentialRenewer** in case of yarn and **MesosHadoopDelegationTokenManager** in case of mesos.

The proposal is to provide obtained delegation tokens to **KafkaConsumer/KafkaProducer** instances using dynamic JAAS configuration if sasl.mechanism is SCRAM related on source/sink (could be set to SCRAM-SHA-256 or SCRAM-SHA-512). Dynamic configuration keeps tokens in-memory which is advisable from a security perspective.

There are cached **KafkaConsumer/KafkaProducer** instances which are considered in the following way:

- **KafkaDataConsumer**: Task could fail because token is changed. If the task is a retry it invalidates the cached consumer and creates a new one with the appropriate token. (This is being rewritten but the mentioned behavior seems stay)
- **CachedKafkaProducer**: This uses the provided kafka parameters as a key so if the token changes a new producer will be created. The old one will evict by default in 10 minutes.

Optional Rejected Designs

Proposal 1:

Instead of defining Kafka parameters in Spark's configuration the existing delegation token framework could be refactored to get started only after source/sink parameters were parsed.

Reject reasons:

- I see high risk to refactor already existing delegation token management for HDFS, Hive and HBase. Honestly I don't think anybody can tell what kind of problems could cause this step.
- It's a huge refactor and I don't see the return of investment.

Proposal 2:

Instead of defining Kafka parameters in Spark's configuration external/kafka-0-10-sql library can build up it's own lazy mechanism to obtain/renew/provide delegation tokens.

Reject reasons:

- There is already a framework to handle delegation tokens and building up a separate universe with maybe copy/paste is rarely a good idea.

Proposal 3:

Instead of defining Kafka parameters in Spark's configuration, a new file can be used.

Reject reasons:

- I’m not sure it is worth it to create a new config for these small amounts of parameters.