Hortonworks Data Platform

Configuring Kafka for Kerberos Over Ambari

(June 28, 2016)

docs.cloudera.com

Hortonworks Data Platform: Configuring Kafka for Kerberos Over Ambari

Copyright © 2012-2016 Hortonworks, Inc. Some rights reserved.

The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including MapReduce, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, ZooKeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, training and partner-enablement services. All of our technology is, and will remain, free and open source.

Please visit the Hortonworks Data Platform page for more information on Hortonworks technology. For more information on Hortonworks services, please visit either the Support or Training page. Feel free to contact us directly to discuss your specific needs.

Except where otherwise noted, this document is licensed under
Creative Commons Attribution ShareAlike 4.0 License.
http://creativecommons.org/licenses/by-sa/4.0/legalcode

Table of Contents

1. Overview

This chapter describes how to configure Kafka for Kerberos security on an Ambari-managed cluster.

Kerberos security for Kafka is an optional feature. When security is enabled, features include:

- Authentication of client connections (consumer, producer) to brokers
- ACL-based authorization

2. Preparing the Cluster

Before you enable Kerberos, your cluster must meet the following prerequisites:

Prerequisite	References*
Ambari-managed cluster with Kafka installed. • Ambari Version 2.1.0.0 or later • Stack version HDP 2.3.2 or later	Installing, Configuring, and Deploying a HDP Cluster in *Automated Install with Ambari*
Key Distribution Center (KDC) server installed and running	Installing and Configuring the KDC
JCE installed on all hosts on the cluster (including the Ambari server)	Enabling Kerberos Authentication Using Ambari

Links are for Ambari 2.2.0.0.

When all prerequisites are fulfilled, enable Kerberos security. (For more information see Launching the Kerberos Wizard (Automated Setup).)

3. Configuring the Kafka Broker for Kerberos

During the installation process, Ambari configures a series of Kafka settings and creates a JAAS configuration file for the Kafka server.

It is not necessary to modify these settings, but for more information see Kafka Configuration Options.

4. Creating Kafka Topics

When you use a script, command, or API to create a topic, an entry is created under ZooKeeper. The only user with access to ZooKeeper is the service account running Kafka (by default, `kafka`). Therefore, the first step toward creating a Kafka topic on a secure cluster is to run `kinit`, specifying the Kafka service keytab. The second step is to create the topic.

1. Run `kinit`, specifying the Kafka service keytab. For example:

   ```
   kinit -k -t /etc/security/keytabs/kafka.service.keytab kafka/
   c6401.ambari.apache.org@EXAMPLE.COM
   ```

2. Next, create the topic. Run the `kafka-topics.sh` command-line tool with the following options:

   ```
   /bin/kafka-topics.sh --zookeeper <hostname>:<port> --create
   --topic <topic-name> --partitions <number-of-partitions> --
   replication-factor <number-of-replicating-servers>
   ```

   For example:

   ```
   /bin/kafka-topics.sh --zookeeper c6401.ambari.apache.org:2181 --create --
   topic test_topic --partitions 2 --replication-factor 2

   Created topic "test_topic".
   ```

   For more information about `kafka-topics.sh` parameters, see Basic Kafka Operations on the Apache Kafka website.

Permissions

By default, permissions are set so that only the Kafka service user has access; no other user can read or write to the new topic. In other words, if your Kafka server is running with principal $KAFKA-USER, only that principal will be able to write to ZooKeeper.

For information about adding permissions, see Authorizing Access when Kerberos is Enabled.

5. Producing Events/Messages to Kafka on a Secured Cluster

Prerequisite: Make sure that you have enabled access to the topic (via Ranger or native ACLs) for the user associated with the producer process. We recommend that you use Ranger to manage permissions. For more information, see the Apache Ranger User Guide for Kafka.

During the installation process, Ambari configures a series of Kafka client and producer settings, and creates a JAAS configuration file for the Kafka client. It is not necessary to modify these settings, but for more information about them see Kafka Configuration Options.

Note: Only the Kafka Java API is supported for Kerberos. Third-party clients are not supported.

To produce events/messages:

1. Specify the path to the JAAS configuration file as one of your JVM parameters:

   ```
   -Djava.security.auth.login.config=/usr/hdp/current/kafka-broker/config/kafka_client_jaas.conf
   ```

   For more information about the `kafka_client_jaas` file, see "JAAS Configuration File for the Kafka Client" in Kafka Configuration Options.

2. `kinit` with the principal's keytab.

3. Launch `kafka-console-producer.sh` with the following configuration options. (Note: these settings are the same as in previous versions, except for the addition of `--security-protocol SASL_PLAINTEXT`.)

   ```
   ./bin/kafka-console-producer.sh --broker-list <hostname:port[,hostname:port, …]> --topic <topic-name> --security-protocol SASL_PLAINTEXT
   ```

   For example:

   ```
   ./bin/kafka-console-producer.sh --broker-list c6401.ambari.apache.org:6667,c6402.ambari.apache.org:6667 --topic test_topic --security-protocol SASL_PLAINTEXT
   ```

Producer Code Example for a Kerberos-Enabled Cluster

The following example shows sample code for a producer in a Kerberos-enabled Kafka cluster. Note that the `SECURITY_PROTOCOL_CONFIG` property is set to `SASL_PLAINTEXT`.

```
package com.hortonworks.example.kafka.producer;

import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.producer.Callback;
```

```
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;

import java.util.Properties;
import java.util.Random;

public class BasicProducerExample {

   public static void main(String[] args){

       Properties props = new Properties();
       props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka.example.
com:6667");

       // specify the protocol for SSL Encryption
       props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG,
 "SASL_PLAINTEXT");

       props.put(ProducerConfig.ACKS_CONFIG, "all");
       props.put(ProducerConfig.RETRIES_CONFIG, 0);
       props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, "org.apache.
kafka.common.serialization.StringSerializer");
       props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, "org.apache.
kafka.common.serialization.StringSerializer");

       Producer<String, String> producer = new KafkaProducer<String,
 String>(props);
       TestCallback callback = new TestCallback();
       Random rnd = new Random();
       for (long i = 0; i < 100 ; i++) {
           ProducerRecord<String, String> data = new ProducerRecord<String,
 String>(
                   "test-topic", "key-" + i, "message-"+i );
           producer.send(data, callback);
       }

       producer.close();
   }


   private static class TestCallback implements Callback {
       @Override
       public void onCompletion(RecordMetadata recordMetadata, Exception e) {
           if (e != null) {
               System.out.println("Error while producing message to topic :" +
 recordMetadata);
               e.printStackTrace();
           } else {
               String message = String.format("sent message to topic:%s
 partition:%s  offset:%s", recordMetadata.topic(), recordMetadata.partition(),
 recordMetadata.offset());
               System.out.println(message);
           }
       }
   }

}
```

To run the example, issue the following command:

```
$ java -Djava.security.auth.login.config=/usr/hdp/current/kafka-broker/
config/kafka_client_jaas.conf com.hortonworks.example.kafka.producer.
BasicProducerExample
```

Troubleshooting

Issue: If you launch the producer from the command-line interface without specifying the `security-protocol` option, you will see the following error:

```
2015-07-21 04:14:06,611] ERROR fetching topic metadata for topics
[Set(test_topic)] from broker
[ArrayBuffer(BrokerEndPoint(0,c6401.ambari.apache.org,6667),
BrokerEndPoint(1,c6402.ambari.apache.org,6667))] failed
(kafka.utils.CoreUtils$)
kafka.common.KafkaException: fetching topic metadata for topics
[Set(test_topic)] from broker
[ArrayBuffer(BrokerEndPoint(0,c6401.ambari.apache.org,6667),
BrokerEndPoint(1,c6402.ambari.apache.org,6667))] failed
     at kafka.client.ClientUtils$.fetchTopicMetadata(ClientUtils.scala:73)
Caused by: java.io.EOFException: Received -1 when reading from channel, socket
 has likely been closed.
     at kafka.utils.CoreUtils$.read(CoreUtils.scala:193)
     at kafka.network.BoundedByteBufferReceive.
readFrom(BoundedByteBufferReceive.scala:54)
```

Solution: Add `--security-protocol SASL_PLAINTEXT` to the `kafka-console-producer.sh` runtime options.

6. Consuming Events/Messages from Kafka on a Secured Cluster

Prerequisite: Make sure that you have enabled access to the topic (via Ranger or native ACLs) for the user associated with the consumer process. We recommend that you use Ranger to manage permissions. For more information, see the Apache Ranger User Guide for Kafka.

During the installation process, Ambari configures a series of Kafka client and producer settings, and creates a JAAS configuration file for the Kafka client. It is not necessary to modify these values, but for more information see Kafka Configuration Options.

Note: Only the Kafka Java API is supported for Kerberos. Third-party clients are not supported.

To consume events/messages:

1. Specify the path to the JAAS configuration file as one of your JVM parameters. For example:

   ```
   -Djava.security.auth.login.config=/usr/hdp/current/kafka-broker/
   config/kafka_client_jaas.conf
   ```

   For more information about the `kafka_client_jaas` file, see "JAAS Configuration File for the Kafka Client" in Kafka Configuration Options.

2. `kinit` with the principal's keytab.

3. Launch `kafka-console-consumer.sh` with the following configuration settings. (Note: these settings are the same as in previous versions, except for the addition of `--security-protocol SASL_PLAINTEXT`.)

   ```
   ./bin/kafka-console-consumer.sh --zookeeper
   c6401.ambari.apache.org:2181 --topic test_topic --from-beginning
   --security-protocol SASL_PLAINTEXT
   ```

Consumer Code Example for a Kerberos-Enabled Cluster

The following example shows sample code for a producer in a Kerberos-enabled Kafka cluster. Note that the `SECURITY_PROTOCOL_CONFIG` property is set to `SASL_PLAINTEXT`.

```
package com.hortonworks.example.kafka.consumer;

import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRebalanceListener;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.TopicPartition;

import java.util.Collection;
```

```
import java.util.Collections;
import java.util.Properties;

public class BasicConsumerExample {

   public static void main(String[] args) {

       Properties consumerConfig = new Properties();
       consumerConfig.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka.
example.com:6667");

       // specify the protocol for SSL Encryption
       consumerConfig.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG,
 "SASL_PLAINTEXT");

       consumerConfig.put(ConsumerConfig.GROUP_ID_CONFIG, "my-group");
       consumerConfig.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
 "earliest");
       consumerConfig.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
 "org.apache.kafka.common.serialization.StringDeserializer");
       consumerConfig.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.
apache.kafka.common.serialization.StringDeserializer");
       KafkaConsumer<byte[], byte[]> consumer = new
 KafkaConsumer<>(consumerConfig);
       TestConsumerRebalanceListener rebalanceListener = new
 TestConsumerRebalanceListener();
       consumer.subscribe(Collections.singletonList("test-topic"),
 rebalanceListener);

       while (true) {
           ConsumerRecords<byte[], byte[]> records = consumer.poll(1000);
           for (ConsumerRecord<byte[], byte[]> record : records) {
               System.out.printf("Received Message topic =%s, partition =%s,
 offset = %d, key = %s, value = %s\n", record.topic(), record.partition(),
 record.offset(), record.key(), record.value());
           }

           consumer.commitSync();
       }

   }

   private static class  TestConsumerRebalanceListener implements
 ConsumerRebalanceListener {
       @Override
       public void onPartitionsRevoked(Collection<TopicPartition> partitions)
 {
           System.out.println("Called onPartitionsRevoked with partitions:" +
 partitions);
       }

       @Override
       public void onPartitionsAssigned(Collection<TopicPartition> partitions)
 {
           System.out.println("Called onPartitionsAssigned with partitions:" +
 partitions);
       }
   }

}
```

To run the example, issue the following command:

```
# java -Djava.security.auth.login.config=/usr/hdp/current/kafka-broker/
config/kafka_client_jaas.conf  com.hortonworks.example.kafka.consumer.
BasicConsumerExample
```

Troubleshooting

Issue: If you launch the consumer from the command-line interface without specifying the `security-protocol` option, you will see the following error:

```
2015-07-21 04:14:06,611] ERROR fetching topic metadata for topics
[Set(test_topic)] from broker
[ArrayBuffer(BrokerEndPoint(0,c6401.ambari.apache.org,6667),
BrokerEndPoint(1,c6402.ambari.apache.org,6667))] failed
(kafka.utils.CoreUtils$)
kafka.common.KafkaException: fetching topic metadata for topics
[Set(test_topic)] from broker
[ArrayBuffer(BrokerEndPoint(0,c6401.ambari.apache.org,6667),
BrokerEndPoint(1,c6402.ambari.apache.org,6667))] failed
     at kafka.client.ClientUtils$.fetchTopicMetadata(ClientUtils.scala:73)
Caused by: java.io.EOFException: Received -1 when reading from channel, socket
 has likely been closed.
     at kafka.utils.CoreUtils$.read(CoreUtils.scala:193)
     at kafka.network.BoundedByteBufferReceive.
readFrom(BoundedByteBufferReceive.scala:54)
```

Solution: Add `--security-protocol SASL_PLAINTEXT` to the `kafka-console-consumer.sh` runtime options.

7. Authorizing Access when Kerberos is Enabled

Kafka ships with a pluggable Authorizer and an out-of-box authorizer implementation that uses ZooKeeper to store Access Control Lists (ACLs). Authorization can be done via Ranger (see the Kafka section of the Ranger Install Guide) or with native ACLs.

A Kafka ACL entry has the following general format:

```
Principal P is [Allowed/Denied] Operation O From Host H On
Resource R
```

where

- A principal is any entity that can be authenticated by the system, such as a user account, a thread or process running in the security context of a user account, or security groups of such accounts. `Principal` is specified in the PrincipalType:PrincipalName (`user:dev@EXAMPLE.COM`) format. Specify user:* to indicate all principals.

  `Principal` is a comma-separated list of principals. Specify * to indicate all principals. (A principal is any entity that can be authenticated by the system, such as a user account, a thread or process running in the security context of a user account, or security groups of such accounts.)

- `Operation` can be one of: `READ`, `WRITE`, `CREATE`, `DESCRIBE`, or `ALL`.

- `Resource` is a topic name, a consumer group name, or the string "kafka-cluster" to indicate a cluster-level resource (only used with a CREATE operation).

- `Host` is the client host IP address. Specify * to indicate all hosts.

Note

For more information about ACL structure, including mappings between Operations values and Kafka protocol APIs, see the Apache KIP-11 Authorization Interface document.

7.1. Kafka Authorization Command Line Interface

The Kafka Authorization CLI script, `kafka-acls.sh`, resides in the `bin` directory.

The following table lists ACL actions supported by the CLI script:

Action Type	Description
`--add`	Add an ACL.
`--remove`	Remove an ACL.
`--list`	List ACLs.

The following table lists additional options for the Authorization CLI:

Option	Description	Default	Option Type
`--authorizer`	The fully-qualified class name of the authorizer.	`kafka.security.auth.SimpleAclAuthorizer`	Configuration
`--authorizer-properties`	A list of key=value pairs that will be passed to authorizer for initialization. Use this option multiple times to specify multiple properties.		Configuration
`--cluster`	Specifies the cluster as resource.		Resource
`--topic <topic-name>`	Specifies the topic as resource.		Resource
`--consumer-group <consumer-group>`	Specifies the consumer group as resource.		Resource

`--allow-principal`	These principals will be used to generate an ACL with `Allow` permission.
	
	Specify principal in `PrincipalType:name` format, such as `user:devadmin`.
	
	To specify more than one principal in a single command, specify this option multiple times. For example:
	
	`--allow-principal user:test1@EXAMPLE.COM`
	
	`--allow-principal user:test2@EXAMPLE.COM`
			Principal

`--deny-principal`	These principals will be used to generate an ACL with Deny permission. Principal is in `PrincipalType:name` format. Multiple principals can be specified (see the `--allow-principal` option).		Principal
`--allow-host`	IP address of the host from which the principals listed in `--allow-principal` will have access. To specify multiple hosts, specify this option multiple times.	if `--allow-principal` is specified, this defaults to *, which translates to "all hosts"	Host
`--deny-host`	IP Address of the host from which the principals listed in `--deny-principal` will be denied access. To specify multiple hosts, specify this option multiple times.	if `--deny-principal` is specified, this defaults to *, which translates to "all hosts"	Host
`--operation`	An operation that will be allowed or denied based on principal options.	All	Operation

Option	Description	Default	Option Type
	Valid values: Read, Write, Create, Delete, Alter, Describe, ClusterAction, All		
`--producer`	Convenience option to add or remove ACLs for the producer role. This will generate ACLs that allow WRITE, DESCRIBE on topic, and CREATE on cluster.		Convenience
`--consumer`	Convenience option to add/remove ACLs for consumer role. This will generate ACLs that allows READ, DESCRIBE on topic, and READ on consumer-group.		Convenience

7.2. Authorization Examples

By default, if a principal does not have an explicit ACL that allows access for an operation to a resource, access requests from the principal will be denied.

The following examples show how to add, remove, and list ACLs.

7.2.1. Grant Read/Write Access to a Topic

To add the following ACL:

"Principals user:bob and user:alice are allowed to perform Operation Read and Write on Topic Test-Topic from Host1 and Host2"

run the CLI with the following options:

```
bin/kafka-acls.sh --add --allow-principal user:bob --allow-
principal user:alice --allow-host host1 --allow-host host2 --
operation Read --operation Write --topic test-topic
```

7.2.2. Grant Full Access to Topic, Cluster, and Consumer Group

To add ACLs to a topic, specify `--topic <topic-name>` as the resource option. Similarly, to add ACLs to cluster, specify `--cluster`; to add ACLs to a consumer group, specify `--consumer-group <group-name>`.

The following examples grant full access for principal `bob` to topic `test-topic` and consumer group `10`, across the cluster. Substitute your own values for principal name, topic name, and group name.

```
bin/kafka-acls.sh --topic test-topic --add --allow-principal
user:bob --operation ALL --config /usr/hdp/current/kafka-broker/
config/server.properties
```

```
bin/kafka-acls.sh --consumer-group 10 --add --allow-principal
user:bob --operation ALL --config /usr/hdp/current/kafka-broker/
config/server.properties
```

```
bin/kafka-acls.sh --cluster --add --allow-principal user:bob --
operation ALL --config /usr/hdp/current/kafka-broker/config/
server.properties
```

7.2.3. Add a Principal as Producer or Consumer

The most common use case for ACL management is to add or remove a principal as producer or consumer. The following convenience options handle these cases.

To add `user:bob` as a producer of Test-topic, run the following command:

```
bin/kafka-acls.sh --add --allow-principal user:bob --producer --
topic test-topic
```

Similarly, to add `user:alice` as a consumer of `test-topic` with consumer group `group-1`, pass the `--consumer` option.

Note

When using the consumer option you must specify the consumer group.

```
bin/kafka-acls.sh --add --allow-principal user:bob --consumer --
topic test-topic --consumer-group group-1
```

7.2.4. Deny Access to a Principal

In rare cases you might want to define an ACL that allows access to all but one or more principals. In this case, use the `--deny-principal` and `--deny-host` options.

For example, to allow all users to read from `test-topic` *except* user `bob` from host `bad-host`:

```
bin/kafka-acls.sh --add --allow-principal user:* --allow-host * --
deny-principal user:bob --deny-host bad-host --operation Read --
topic test-topic
```

7.2.5. Remove Access

Removing ACLs is similar to adding ACLs. The only difference is that you need to specify the `--remove` option instead of the `--add` option.

To remove the ACLs for principals bob and alice (added in "Grant Read/Write Access to a Topic"), run the CLI with the following options:

```
bin/kafka-acls.sh --remove --allow-principal user:bob --allow-
principal user:alice --allow-host host1 --allow-host host2 --
operation Read --operation Write --topic test-topic
```

Similarly, to remove a principal from a producer or consumer role, specify the `--remove` option instead of `--add`:

```
bin/kafka-acls.sh --remove --allow-principal user:bob --producer
--topic test-topic
```

7.2.6. List ACLs

To list ACLs for any resource, specify the `--list` option with the resource. For example, to list all ACLs for Test-topic, run the CLI with following options:

```
bin/kafka-acls.sh --list --topic test-topic
```

7.2.7. Configure Authorizer Settings

To specify which authorizer to use, include the `--authorizer` option. For example:

```
--authorizer kafka.security.auth.SimpleAclAuthorizer ...
```

To specify one or more authorizer initialization settings, include the –authorizer-properties option; for example:

```
--authorizer-properties zookeeper.connect=localhost:2181 ...
```

7.3. Troubleshooting Authorizer Settings

Frequently-asked Questions:

When should I use Deny?

By default, all principals that are not explicitly granted permissions get rejected. You should not need to use Deny. (Note: when defined, DENY takes precedence over ALLOW.)

Then why do we have deny?

Deny was introduced into Kafka for advanced use cases where negation was required. Deny should only be used to negate a large allow, where listing all principals or hosts is cumbersome.

Can I define ACLs with principal as user@<realm>?

You can if you are not using `principal.to.local.class`, but if you have set this configuration property you must define your ACL with users without REALM. This is a known issue in HDP 2.3.

I just gave a user CREATE Permissions on a cluster, but the user still can't create topics. Why?

Right now, Kafka create topic is not implemented as an API, but as a script that directly modifies ZooKeeper entries. In a secure environment only the Kafka broker user is allowed to write to ZooKeeper entries. Granting a user CREATE access does not allow that user to modify ZooKeeper entries.

However, if that user makes a producer request to the topic and has `auto.create.topics.enable` set to `true`, a topic will be created at the broker level.

8. Appendix: Kafka Configuration Options

8.1. Server.properties key-value pairs

Ambari configures the following Kafka values during the installation process. Settings are stored as key-value pairs stored in an underlying `server.properties` configuration file.

listeners

A comma-separated list of URIs that Kafka will listen on, and their protocols.

Required property with three parts:

`<protocol>:<hostname>:<port>`

Set `<protocol>` to SASL_PLAINTEXT, to specify the protocol that server accepts connections. SASL authentication will be used over a plaintext channel. Once SASL authentication is established between client and server, the session will have the client's principal as an authenticated user. The broker can only accept SASL (Kerberos) connections, and there is no wire encryption applied. (Note: For a non-secure cluster, `<protocol>` should be set to PLAINTEXT.)

Set `hostname` to the hostname associated with the node you are installing. Kerberos uses this value and "principal" to construct the Kerberos service name. Specify hostname `0.0.0.0` to bind to all interfaces. Leave hostname empty to bind to the default interface.

Set `port` to the Kafka service port. When Kafka is installed using Ambari, the default port number is 6667.

Examples of legal listener lists::

```
listeners=SASL_PLAINTEXT://kafka1.host1.com:6667
```

```
listeners=PLAINTEXT://myhost:9092, TRACE://:9091,
SASL_PLAINTEXT://0.0.0.0:9093
```

advertised.listeners

A list of listeners to publish to ZooKeeper for clients to use, if different than the listeners specified in the preceding section.

In IaaS environments, this value might need to be different from the interface to which the broker binds.

If `advertised.listeners` is not set, the value for `listeners` will be used.

Required value with three parts:

`<protocol>:<hostname>:<port>`

Set `protocol` to SASL_PLAINTEXT, to specify the protocol that server accepts connections. SASL authentication will be used over a plaintext channel. Once SASL authentication is established between client and server, the session will have the client's principal as an authenticated user. The broker can only accept SASL (Kerberos) connections, and there is no wire encryption applied. (Note: For a non-secure cluster, `<protocol>` should be set to PLAINTEXT.)

Set `hostname` to the hostname associated with the node you are installing. Kerberos uses this and "principal" to construct the Kerberos service name.

Set `port` to the Kafka service port. When Kafka is installed using Ambari, the default port number is 6667.

For example:

```
advertised.listeners=SASL_PLAINTEXT://kafka1.host1.com:6667
```

security.inter.broker.protocol

Specifies the inter-broker communication protocol. In a Kerberized cluster, brokers are required to communicate over SASL. (This approach supports replication of topic data.) Set the value to SASL_PLAINTEXT:

```
security.inter.broker.protocol=SASL_PLAINTEXT
```

authorizer.class.name

Configures the authorizer class.

Set this value to `kafka.security.auth.SimpleAclAuthorizer`:

```
authorizer.class.name=kafka.security.auth.SimpleAclAuthorizer
```

For more information, see "Authorizing Access when Kerberos is Enabled."

principal.to.local.class

Transforms Kerberos principals to their local Unix usernames.

Set this value to `kafka.security.auth.KerberosPrincipalToLocal`:

```
principal.to.local.class=kafka.security.auth.KerberosPrincipalToLocal
```

super.users

Specifies a list of user accounts that will have all cluster permissions. By default, these super users have all permissions that would otherwise need to be added through the `kafka-acls.sh` script. Note, however, that their permissions do not include the ability to create topics through `kafka-topics.sh`, as this involves direct interaction with ZooKeeper.

Set this value to a list of `user:<account>` pairs separated by semicolons. Note that Ambari adds `user:kafka` when Kerberos is enabled.

Here is an example:

```
super.users=user:bob;user:alice
```

8.2. JAAS Configuration File for the Kafka Server

The Java Authentication and Authorization Service (JAAS) API supplies user authentication and authorization services for Java applications.

After enabling Kerberos, Ambari sets up a JAAS login configuration file for the Kafka server. This file is used to authenticate the Kafka broker against Kerberos. The file is stored at:

```
/usr/hdp/current/kafka-broker/config/kafka_server_jaas.conf
```

Ambari adds the following settings to the file. (Note: serviceName="kafka" is required for connections from other brokers.)

```
KafkaServer {
       com.sun.security.auth.module.Krb5LoginModule required
       useKeyTab=true
       keyTab="/etc/security/keytabs/kafka.service.keytab"
       storeKey=true
       useTicketCache=false
       serviceName="kafka"
       principal="kafka/c6401.ambari.apache.org@EXAMPLE.COM";
       };

Client { // used for zookeeper connection
       com.sun.security.auth.module.Krb5LoginModule required
       useKeyTab=true
       keyTab="/etc/security/keytabs/kafka.service.keytab"
       storeKey=true
       useTicketCache=false
       serviceName="zookeeper"
       principal="kafka/c6401.ambari.apache.org@EXAMPLE.COM";
       };
```

8.3. Configuration Setting for the Kafka Producer

After enabling Kerberos, Ambari sets the following key-value pair in the `server.properties` file:

```
security.protocol=SASL_PLAINTEXT
```

8.4. JAAS Configuration File for the Kafka Client

After enabling Kerberos, Ambari sets up a JAAS login configuration file for the Kafka client. Settings in this file will be used for any client (consumer, producer) that connects to a Kerberos-enabled Kafka cluster. The file is stored at:

`/usr/hdp/current/kafka-broker/config/kafka_client_jaas.conf`

Ambari adds the following settings to the file. (Note: `serviceName=kafka` is required for connections from other brokers.)

Note

For command-line utilities like kafka-console-producer and kafka-console-consumer, use `kinit`. If you use a long-running process (for example, your own Producer), use keytab.

Kafka client configuration *with* keytab, for *producers*:

```
KafkaClient {
     com.sun.security.auth.module.Krb5LoginModule required
     useKeyTab=true
     keyTab="/etc/security/keytabs/storm.service.keytab"
     storeKey=true
     useTicketCache=false
     serviceName="kafka"
     principal="storm@EXAMPLE.COM";
    };
```

Kafka client configuration *without* keytab, for *producers*:

```
KafkaClient {
     com.sun.security.auth.module.Krb5LoginModule required
     useTicketCache=true
     renewTicket=true
     serviceName="kafka";
    };
```

Kafka client configuration for *consumers*:

```
 KafkaClient {
     com.sun.security.auth.module.Krb5LoginModule required
     useTicketCache=true
     renewTicket=true
     serviceName="kafka";
     };
```