Kafka 学习笔记

Kafka 安全

概要

- 安全总览
- 使用SSL加密和授权

安全总览

在0.9.0.0版中，Kafka社区添加了一些特性，通过单独使用或者一起使用这些特性，提高了Kafka集群的安全性。目前支持下列安全措施。

- 使用SSL或SASL验证来自客户端(producers和consumers)、其他brokers和工具的连接。Kafka支持以下SASL机制：
  - SASL/GSSAPI (Kerberos) - 从版本0.9.0.0开始
  - SASL/PLAIN - 从版本0.10.0.0开始
  - SASL/SCRAM-SHA-256 和 SASL/SCRAM-SHA-512 - 从版本0.10.2.0开始
- 验证从brokers 到 ZooKeeper的连接。
- 对brokers与clients之间、brokers之间或brokers与工具之间使用SSL传输对数据加密(注意，启用SSL时性能会下降，其大小取决于CPU类型和JVM实现)。
- 授权客户端的读写操作
- 授权是可插拔的，并且支持与外部授权服务的集成

> 值得注意的是，安全是可选的 - 支持非安全集群，也支持需要认证，不需要认证，加密和未加密clients的混合集群。 以下指南介绍了如何在clients和brokers中配置和使用安全功能。

使用SSL加密和授权

Apache Kafka允许客户端通过SSL进行连接。默认情况下，SSL被禁用，但可以根据需要启用。

为每个Kafka broker生成SSL密钥和证书

部署一个或多个支持SSL的brokers的第一步是为集群中的每台计算机生成密钥和证书。你可以使用Java的keytool实用程序来完成此任务。最初我们将生成一个临时密钥库的密钥，以便我们稍后可以导出并用CA签名。

```
keytool -keystore server.keystore.jks -alias localhost -validity {validity} -genkey -keyalg RSA
```

你需要在上面的命令中指定两个参数：

- keystore: 存储证书的密钥库文件。密钥库文件包含证书的私钥；因此，它需要安全保存。
- validity: 证书有效期。
  注意：默认情况下，属性 `ssl.endpoint.identification.algorithm` 未定义，所以不执行主机名验证。为了启用主机名验证，请设置以下属性：

```
ssl.endpoint.identification.algorithm=HTTPS
```

启用后，客户端将根据以下两个字段之一验证服务器的完全限定域名(FQDN)：

- Common Name (CN)
- Subject Alternative Name (SAN)

这两个字段都是有效的，但RFC-2818建议使用SAN。 SAN也更加灵活，允许声明多个DNS条目。另一个优点是可以将CN设置为更有意义的值用于授权目的。要添加SAN字段，请将以下参数 `-ext SAN=DNS:{FQDN}` 附加到keytool命令中：

```
keytool -keystore server.keystore.jks -alias localhost -validity {validity} -genkey -keyalg RSA
```

之后可以运行以下命令来验证生成的证书内容：

```
keytool -list -v -keystore server.keystore.jks
```

创建你自己的证书管理机构 (CA)

在完成第一步之后, 集群中的每一台机器都有一个公私密钥对, 和一个用于标识该机器的证书. 但是, 这个证书是未签名的, 也就是说攻击者也可以创造一个这样的证书, 假装是其中任何一台机器.
因此, 给集群中每个机器的证书签名, 来防止伪造的证书, 是非常重要的. 一个证书管理机构 (CA) 负责证书的签名. CA 的工作流程类似政府发放护照. 政府给每一个护照盖章(签名), 因此护照变得难以伪造. 其他政府通过验证这个盖章来确认护照是可信的. 同理, CA 给证书签名, 并且加密方法保证已经签名的证书是难以计算和伪造的. 所以, 只要 CA 是真实的可信的权威机构, 客户端能更好的保证他们连接到的是可信的机器.

```
openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
```

生成的 CA 是一个简单的公私密钥对和证书, 然后, 它将对其他证书签名.
下一步是将生成的 CA 添加到 **客户端的信任存储区 (clients' truststore)**, 这样客户端才可以相信这个

CA:

```
keytool -keystore client.truststore.jks -alias CARoot -import -file ca-cert
```

Note: 如果你将 Kafka broker 配置成需要客户端授权, 可以在 Kafka broker 配置中设置 `setting ssl.client.auth` 属性为 "requested" 或者 "required". 然后你也一定要给 Kafka broker 提供一个信任存储区, 并且这个信任存储区应该拥有所有给客户端密钥签名的 CA 证书.

```
keytool -keystore server.truststore.jks -alias CARoot -import -file ca-cert
```

相对于第一步中的密钥库 (keystore) 存储每个机器自己的标识, 客户端的信任存储区 (truststore) 保存所有客户端需要信任的证书. 导入一个证书到一台机器的信任存储区, 意味着这台机器将信任所有被这个证书签名的证书. 与上面相似的, 相信政府 (CA) 即相信政府发放的所有护照(证书). 这一特征称为信任链, 当在大型的 Kafka 集群中部署 SSL 时, 这一特征特别有用. 你可以用一个单一的 CA 给集群中的所有证书签名, 而且可以让所有机器共享信任该 CA 的信任存储区. 这样, 每个机器都可以认证除自己之外的全部机器.

给证书签名

下一步是用第二步生成的 CA 给第一步生成的所有证书签名. 首先, 你需要从密钥库中把证书导出来:

```
keytool -keystore server.keystore.jks -alias localhost -certreq -file cert-file
```

然后用 CA 给导出的证书签名:

```
openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days {validity} -CAc
```

最后, 你要把 CA 的证书和被签名的证书一起导入密钥库中:

```
keytool -keystore server.keystore.jks -alias CARoot -import -file ca-cert
keytool -keystore server.keystore.jks -alias localhost -import -file cert-signed
```

参数的定义如下所示:

- keystore: 密钥库的地址
- ca-cert: CA 的证书
- ca-key: CA 的私钥
- ca-password: CA 的密码
- cert-file: 从服务器导出的未被签名的证书

- cert-signed: 已经签名的服务器的证书

这是一个集合上面所有步骤, 用 bash 写的例子. 注意其中一条命令假设密码是 `test1234` , 所以你要么使用这个密码, 要么在执行命令前修改该密码:

```
#!/bin/bash
#Step 1
keytool -keystore server.keystore.jks -alias localhost -validity 365 -keyalg RSA -genkey
#Step 2
openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
keytool -keystore server.truststore.jks -alias CARoot -import -file ca-cert
keytool -keystore client.truststore.jks -alias CARoot -import -file ca-cert
#Step 3
keytool -keystore server.keystore.jks -alias localhost -certreq -file cert-file
openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days 365 -CAcreate
keytool -keystore server.keystore.jks -alias CARoot -import -file ca-cert
keytool -keystore server.keystore.jks -alias localhost -import -file cert-signed
```

配置 Kafka Broker

Kafka Broker 支持在多个端口上监听连接. 我们需要在 server.properties 中配置以下属性, 必须有一个或多个用逗号隔开的值:

```
listeners
```

如果 broker 之间的通信没有启用 SSL (如何启用可参照下文). PLAINTEXT 和 SSL 两个协议都需要提供端口信息.

```
listeners=PLAINTEXT://host.name:port,SSL://host.name:port
```

下面是 broker 端需要的 SSL 的配置:

```
ssl.keystore.location=/var/private/ssl/server.keystore.jks
ssl.keystore.password=test1234
ssl.key.password=test1234
ssl.truststore.location=/var/private/ssl/server.truststore.jks
ssl.truststore.password=test1234
```

Note: 严格来讲 ssl.truststore.password 是可选配置的, 但是强烈建议配置. 如果没有配置 password, 依然可以访问信任存储区 (truststore), 但是不能进行真实性检查. 值得考虑的可选配置:

- ssl.client.auth=none ("required" => 客户端授权是必须的, "requested" => 客户端授权是需要的, 但是没有证书依然可以连接. 因为 "requested" 会造成错误的安全感, 而且在客户端配置错误的情况下依然可以连接成功, 所以不鼓励使用.)

- ssl.cipher.suites (可选的). 指定的密码套件, 由授权, 加密, MAC 和密钥交换算法组成, 用于给使用 TLS 或者 SSL 协议的网络协商安全设置. (默认是空的 list)
- ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1 (列出你将要从客户端接收的 SSL 协议. 注意, SSL 已经废弃, 支持 TLS, 并且不建议在生产环境中使用 SSL.)
- ssl.keystore.type=JKS
- ssl.truststore.type=JKS
- ssl.secure.random.implementation=SHA1PRNG

如果你想 broker 间的通信启用 SSL, 将下面内容添加到 server.properties 文件中(默认是 PLAINTEXT):

```
security.inter.broker.protocol=SSL
```

由于一些国家的进口规定, Oracle 的实现限制了默认加密算法的强度. 如果需要更强的算法 (例如, 256 位的 AES 密钥), 必须获得 JCE Unlimited Strength Jurisdiction Policy 文件, 并且在 JDK/JRE 中安装. 参考 JCA Providers 文档 获取更多新.

JRE/JDK 将有一个默认的伪随机数生成器 (PRNG), 用于加密操作, 所以没有要求使用

```
ssl.secure.random.implementation
```

配置 PRNG 的实现. 但是, 其中某些实现会造成性能问题 (尤其是, Linux 系统的默认选择,
`NativePRNG`
, 利用了一个全局锁.) 万一 SSL 连接的性能变成一个问题, 可以考虑, 明确的设置要使用的 PRNG 实现.
`SHA1PRNG`
实现是非阻塞的, 在高负载 (加上复制消息的流量, 每个 broker 生产消息的速度是 50 MB/sec) 下有非常好的性能表现.
一旦你启动了 broker, 你应该能在 server.log 中看到如下内容:

```
with addresses: PLAINTEXT -> EndPoint(192.168.64.1,9092,PLAINTEXT),SSL -> EndPoint(192.168.64.1,
```

用以下命令可以快速验证服务器的 keystore 和 truststore 是否设置正确:

```
openssl s_client -debug -connect localhost:9093 -tls1
```

Note: TLSv1 需要在 ssl.enabled.protocols 中列出)
在命令的输出中, 你应该能看到服务器的证书:

```
   -----BEGIN CERTIFICATE-----
{variable sized random bytes}
-----END CERTIFICATE-----
subject=/C=US/ST=CA/L=Santa Clara/O=org/OU=org/CN=Sriharsha Chintalapani
issuer=/C=US/ST=CA/L=Santa Clara/O=org/OU=org/CN=kafka/emailAddress=test@test.com
```

如果没有显示证书信息, 或者有任何其他错误信息, 那么你的 keystore 设置不正确。

配置 Kafka 客户端

只有新的 Kafka Producer 和 Consumer 支持 SSL, 旧的 API 不支持 SSL. Producer 和 Consumer 的 SSL 的配置相同.
如果在 broker 中不需要客户端授权, 那么下面是最小的配置的例子:

```
security.protocol=SSL
ssl.truststore.location=/var/private/ssl/client.truststore.jks
ssl.truststore.password=test1234
```

Note: 严格来讲 `ssl.truststore.password` 是可选配置的, 但是强烈建议配置. 如果没有配置 password, 依然可以访问 truststore, 但是不能进行真实性检查. 如果要求客户端授权, 必须像第一步一样创建一个 keystore, 和配置下面这些信息:

```
ssl.keystore.location=/var/private/ssl/client.keystore.jks
ssl.keystore.password=test1234
ssl.key.password=test1234
```

根据我们的需求和 broker 的配置, 也需要设置其他的配置:

- `ssl.provider` (可选的). 用于 SSL 连接的安全提供程序名称. 默认值是 JVM 的默认安全提供程序.
- `ssl.cipher.suites` (可选的). 指定的密码套件, 由授权, 加密, MAC 和密钥交换算法组成, 用于给使用 TLS 或者 SSL 协议的网络协商安全设置.
- `ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1` . 需要列出至少一个在 broker 端配置的协议
- `ssl.truststore.type=JKS`
- `ssl.keystore.type=JKS`

使用 console-producer 和 console-consumer 的例子:

```
kafka-console-producer.sh --broker-list localhost:9093 --topic test --producer.config client-ssl
kafka-console-consumer.sh --bootstrap-server localhost:9093 --topic test --consumer.config clier
```